Improve Reasoning Ability by Reinforcing Only from Positive Rollouts

Mingwei Xu [*1] **Hao Fang** [*1]

Abstract

Distillation and reinforcement learning through verifiable rewards (RLVR) achieved progress for enhancing the reasoning ability of large language models (LLMs). However, we note that under sparse binary rewards, negative rollouts admit no gradation of failure severity, and the combinatorial vastness makes penalizing a few sampled negatives unlikely to cover a meaningful gradient signal. In this work, we propose **Positive-Only Policy Optimization (POPO)**, a self-distillation integrated RLVR framework in which learning occurs exclusively on online positive rollouts. Specifically, POPO utilizes bounded importance sampling over the positive rollout set. Thus, no disjoint negative rollouts are used for the gradient guidance during the policy post-training. We show that implicit negative gradients can emerge naturally through reinforcing the positive probability via rollouts redistribution. Next, POPO stabilizes the policy optimization through self-distillation. First, it applies a siamese policy network with a momentum-based adaptation law for policy evolution. Second, we replace the KL-divergence with a bounded similarity penalty term in the siamese representation space.We conduct experiments using public text LLMs across mathematical benchmarks and find that POPO matches or even surpasses GRPO. Notably, we show that POPO can achieve 36.67% in AIME 2025 with Qwen-Math-7B, outperforming GRPO 30.00%. Our ablation and sweep studies further illustrate the necessity and robustness of POPO components. Github: https://github.com/momo1443/self-distillation-rlvr-popo.

[1]Department of Electrical and Computer Engineering, University of Washington, Seattle, WA 98195, USA. Correspondence to: Hao Fang <haof459@uw.edu>.

Accepted at 43rd International Conference on Machine Learning (ICML) 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning, Seoul, South Korea

1. Introduction

The emergence of reasoning capabilities in post-training of large language models (LLMs) through reinforcement learning (RL) makes a significant advance in the capability enhancement and human feedback (RLHF) alignment (Ouyang et al., 2022). Well-known work, such as DeepSeek-R1 (Guo et al., 2025) and OpenAI's o1 series (Jaech et al., 2024), demonstrates that LLMs can surge intelligent chain-of-thought (COT) (Wei et al., 2022) reasoning ability through RL with Verifiable Rewards (RLVR) (Lambert et al., 2024). In general, RLVR works by giving a sampled question x and the corresponding LLM-generated response y containing reasoning steps and a final answer, a verifier provides deterministic feedback (Cobbe et al., 2021). The simplicity of the verification mechanism, together with various RL algorithms, has achieved substantial improvements in mathematical problem-solving and coding tasks (Le et al., 2022), where LLMs show a strong ability of self-reflection and generating sophisticated reasoning chains.

Despite these successes, current RLVR approaches face a fundamental challenge: the treatment of incorrect responses. Take the mathematical reasoning task, for example. At the cold-start stage, the vast majority of sampled responses are incorrect, e.g., typical pass@k rates are usually below 50% for the AMC 23 dataset, which happens due to diverse situations such as computational errors, problem misinterpretations, and inappropriate solution strategies. Thus, we begin to reconsider and ask the question: *Can a policy also improve by reinforcing only from its successes?* This seems counterintuitive at first glance, given the conventions in RL, which reinforce the policy with the good reward while penalizing it for the bad rewards. However, a few insightful observations motivate it. First, the space of incorrect responses is combinatorially vast (Yue et al., 2025) for mathematical tasks, making it unlikely that penalizing a few sampled failures meaningfully covers the failure modes. Second, on-policy distillation (OPD) (Agarwal et al., 2024; Gu et al., 2024) and self-distillation (Mobahi et al., 2020) are shown to be effective as an alternative post-training approach. Third, past work in contrastive self-supervised learning (SSL) demonstrates that effective representations are feasible using only positive samples, works such as "Bootstrap your own latent" (BYOL) (Grill et al., 2020) and "Exploring

simple siamese representation learning"(SimSiam) (Chen & He, 2021), achieving powerful latent representations.

Inspired by the above, we propose **Positive-Only Policy Optimization (POPO)**, a self-distillation integrated on-policy optimization approach that learns exclusively on positive reward responses. Specifically, POPO has the following contributions:

- POPO introduces bounded importance sampling weights, which normalize over the positive rollout set, to preferentially reinforce positive responses, while introducing implicit negative penalties.
- POPO uses an exponential moving average (EMA) to update the siamese policy network as an adaptive policy anchor, combining the benefits from SSL and distillation insights.
- POPO replaces KL divergence with representation-space similarity, providing a soft bounded constraint that preserves semantic structure in the predictor space.

To evaluate the POPO algorithm, we use the DeepScaleR-Preview-Dataset for policy training. Then, we conduct extensive test experiments using publicly available, well-established LLM models, including the Qwen2.5 Math family, DeepSeek-R1 distilled models, and general text-LLM models, across all-level mathematical reasoning benchmarks (MATH-500, AMC23, AIME 2024, AIME 2025, and Olympiad). Our experiment demonstrates that POPO still achieves comparable or even superior performance compared with GRPO and its variants. Especially, in AIME 2025, we show that POPO can achieve 36.67% using Qwen-Math-7B, outperforming GRPO 30.00%. Last, our ablation and sweep studies illustrate the necessity and robustness of the components and hyperparameters.

2. Methods

2.1. Problem Setting and Preliminary

We are given a prompt x, the policy π_θ generates a response $y = (y_1, \ldots, y_T)$ autoregressively, and a verifier provides sparse binary feedback $R(x, y) = \mathbf{1}[\texttt{extract}(y) = a^*]$ (Cobbe et al., 2021). We partition rollouts into a positive set $\mathcal{S}^+(x) = \{y : R(x, y) = 1\}$ and a negative set $\mathcal{S}^-(x) = \{y : R(x, y) = 0\}$. Standard policy gradient methods reinforce responses with positive advantage while penalizing negative ones. POPO departs from this convention by optimizing exclusively over $\mathcal{S}^+(x)$.

2.2. Positive-Only Policy Optimization

The overall diagram and algorithm can be found in Figure 1 and Algorithm 1. The overall loss is:

$$\mathcal{L}_{\text{POPO}}(\theta) = -\mathbb{E}_{x\sim\mathcal{D}}\left[\sum_{y\in\mathcal{S}^+(x)} w_\theta(y \mid x)\cdot\log\pi_\theta(y \mid x)\right] + \alpha\mathcal{L}_{\text{sim}}(\theta,\phi,\xi) + \beta\mathcal{L}_{\text{ent}}(\theta), \tag{1}$$

where $w_\theta(y \mid x)$ is a self-normalized importance weight over the positive set, $\mathcal{L}_{\text{sim}}$ provides representation-space regularization, $\mathcal{L}_{\text{ent}}$ is an entropy bonus (Schulman et al., 2017), and $\alpha, \beta > 0$ control regularization strength. The key distinction is that the expectation is taken *only* over $\mathcal{S}^+(x)$. Although negative responses receive no explicit gradient signal, we show in section 2.3 that implicit negative gradients emerge naturally.

Self-Normalized Weight over Positive Set. We define $w_\theta(y \mid x) = \pi_\theta(y \mid x)/Z^+(x)$, where $Z^+(x) = \sum_{y'\in\mathcal{S}^+(x)}\pi_\theta(y' \mid x)$. This creates a *self-competition* mechanism: responses assigned higher probability receive larger weights, reinforcing confident correct answers while maintaining diversity through normalization.

Adaptive Anchor. Methods like BYOL (Grill et al., 2020) and SimSiam (Chen & He, 2021) achieve representation learning using only positive samples. we maintain two copies of the policy: the policy network π_θ with parameters θ that we optimize, and a siamese policy π_ξ with parameters ξ that provides a stable anchor. Siamese policy parameters are updated using an exponential moving average (EMA) of the policy network parameter:$\xi \leftarrow \tau\cdot\xi + (1-\tau)\cdot\theta$, where $\tau \in (0, 1)$ is the momentum coefficient, e.g., $\tau = 0.999$.

Representation-Space Alignment. We replace KL divergence with cosine similarity between hidden representations of the two networks:

$$\mathcal{L}_{\text{sim}}(\theta,\phi,\xi) = -\mathbb{E}_{x\sim\mathcal{D}}\left[\sum_{y\in\mathcal{S}^+(x)} w_\theta(y \mid x)\cdot\cos(\cdot,\cdot)\right], \tag{2}$$

where $\cos(\cdot,\cdot) = \cos\big(h_\phi(f_\theta(x,y)),\ \text{sg}(f_\xi(x,y)+\epsilon)\big)$; $f_\theta(x,y)$ and $f_\xi(x,y)$ are representations extracted from the both networks, respectively; h_ϕ is a predictor head (implemented as MLPs), $\text{sg}(\cdot)$ denotes stop-gradient, and $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ is Gaussian noise. Specifically, **Predictor head.** Predictor h_ϕ on the policy network is an asymmetrical design to ensure that the policy network must do "extra work" to match the Siamese network, preventing the collapse where both networks converge to outputs; **Stop-gradient.** The $\text{sg}(\cdot)$ operator ensures gradients flow only through the policy network. The Siamese network is updated solely via EMA; **Gaussian noise.** Adding small noise

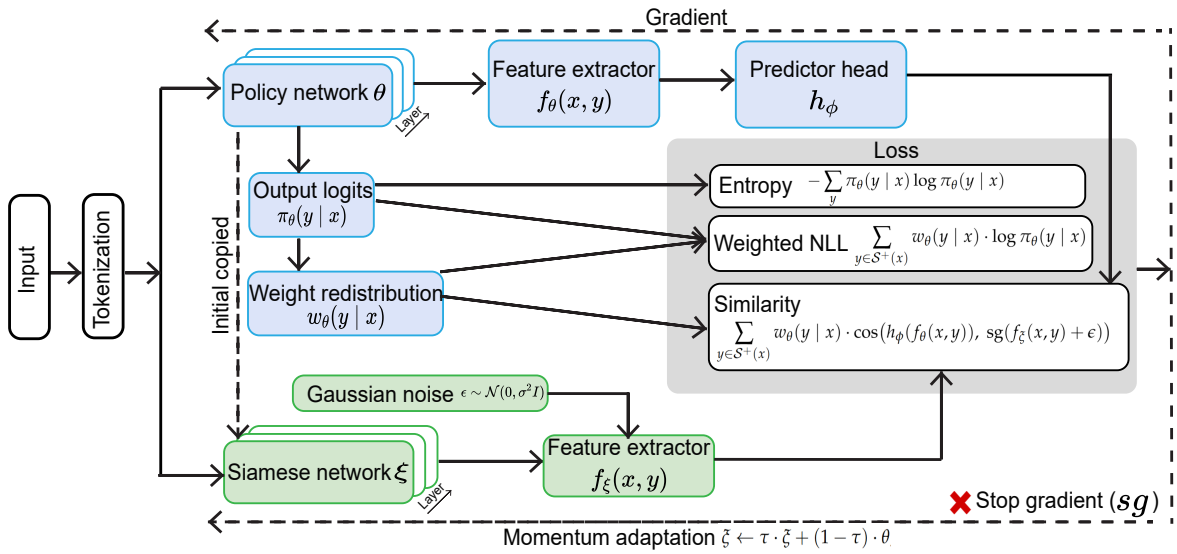

Figure 1. Overview of the POPO components. The policy network (upper part) follows the gradient backpropagation, while the Siamese network (lower part) uses momentum adaptation. Loss is in the grey box. Algorithm 1 shows the forward and backward processes.

ϵ provides robustness against trivial embeddings.

Entropy Regularization. Like the recent approach (Schulman et al., 2017), our POPO also includes the entropy loss to ensure diversity:

$$\mathcal{L}_{\text{ent}}(\theta) = -\mathbb{E}_{x \sim \mathcal{D}} \left[H(\pi_\theta(\cdot \mid x)) \right]. \tag{3}$$

where $H(\pi_\theta(\cdot \mid x)) = -\sum_y \pi_\theta(y \mid x) \log \pi_\theta(y \mid x)$.

2.3. Learning via Implicit Negative Reward

We show that POPO implicitly suppresses incorrect responses without explicit penalties. The fundamental constraint $\sum_y \pi_\theta(y \mid x) = 1$ implies that reinforcing $\pi_\theta(y \mid x)$ for $y \in \mathcal{S}^+(x)$ forces $\pi_\theta(y' \mid x)$ to decrease for $y' \in \mathcal{S}^-(x)$.

Theorem 2.1 (Implicit Negative Gradient)**.** *For any incorrect response $y' \in \mathcal{S}^-(x)$, the gradient of $\mathcal{L}_{POPO}$ (1) with respect to the logit $z_{y'}$ satisfies*

$$\frac{\partial \mathcal{L}_{POPO}}{\partial z_{y'}} = \pi_\theta(y' \mid x) \Big[1 + \beta \Big(\log \pi_\theta(y' \mid x) + H(\pi_\theta(\cdot \mid x)) \Big) \Big]. \tag{4}$$

which is strictly positive for any incorrect response with non-negligible probability. Intuitively, two forces drive this: a uniform "probability tax" from the NLL loss on positives (via the softmax partition function), and a probability-dependent penalty from entropy regularization that is strongest on the most probable incorrect responses. The full proof is given in Appendix B.

3. Experiments

3.1. Experimental Setup

Training Data and Evaluation Benchmarks. We adopt the DeepScaleR-Preview-Dataset (Luo et al., 2025) as the *training* dataset for all experiments. Each problem provides a ground-truth answer for binary reward verification: $R(x, y) = \mathbf{1}[\texttt{extract}(y) = a^*]$. We evaluate on the five benchmarks: MATH-500 (Hendrycks et al., 2021), AMC 2023 (Li et al., 2024), AIME 2024 (Li et al., 2024), AIME 2025 (Balunović et al., 2025), OlympiadBench (He et al., 2024). **Base Models and Algorithm Baselines.** We conduct full-parameter fine-tuning across four base models spanning two scales ($\sim$1.5B and $\sim$7B) and two pretraining paradigms. Math-specialized: Qwen2.5-Math-1.5B and Qwen2.5-Math-7B (Yang et al., 2024). Distilled-reasoning: DeepSeek-R1-Distill-Qwen-1.5B (named as R1-Distill-1.5B) (Guo et al., 2025). All methods are implemented using the *Hugging Face TRL* package for reward computation, training, and generation. We compare four methods: Base, SFT (Liu et al., 2023), GRPO (Shao et al., 2024), and POPO. **Evaluation Protocol.** We consistently report the Pass@8 metrics for all benchmarks. Pass@k is computed using the unbiased estimator (Chen et al., 2021; Lewkowycz et al., 2022; Wang et al., 2022). Output responses use vLLM (Kwon et al., 2023) for fast generation.

3.2. POPO achieved superior performance across models and benchmarks

We began our experimental analysis with a small yet effective Qwen-Math-1.5B model, which has been widely used in recent studies (Wang et al., 2025). In Table 1 first row, we

Table 1. Comparison of RL algorithms' performance across mathematical reasoning benchmarks using math-specialized and distilled base models. Bold numbers indicate the best.

Models	Algorithms	**MATH-500**	**AMC 23**	**AIME 24**	**AIME 25**	**Olympiad**	**Avg.**
Qwen-Math-1.5B	Base	80.20	65.00	20.00	6.67	48.22	44.02
	SFT	78.80	71.89	20.70	10.00	47.77	45.83
	GRPO	86.20	75.00	23.33	16.25	50.30	50.22
	POPO (our)	**86.60**	**77.50**	**26.67**	**23.33**	**51.19**	**53.06**
R1-Distill-1.5B	Base	86.20	75.00	20.00	20.00	54.15	51.07
	SFT	87.00	77.08	21.07	21.19	55.04	52.28
	GRPO	90.20	80.00	30.00	26.32	58.61	57.03
	POPO (our)	**90.80**	**85.24**	**36.67**	**27.86**	59.05	**59.92**
Qwen-Math-7B	Base	82.60	72.50	33.33	20.00	44.21	50.53
	SFT	84.00	70.55	32.81	18.93	44.07	50.07
	GRPO	**90.80**	85.50	43.33	30.00	56.38	61.20
	POPO (our)	**90.80**	**85.75**	**45.13**	**36.67**	**57.27**	**63.12**

observed that the Qwen-Math-1.5B model achieved superior performance across all five mathematical benchmarks (MATH-500: 86.60, AMC 23: 77.50, AIME 24: 26.67, AIME 25: 23.33, Olympiad: 51.19). Further, compared to the other baseline algorithms, our POPO performed in average better across five datasets (POPO: 53.06, v.s. GRPO: 50.22; SFT: 45.83). These results gave the first confidence that our proposed POPO also stabilized reasonable policy optimization using only online positive rollouts.

Next, we further tested our POPO using R1-Distill-1.5B, Qwen-Math-7B (see Table 1, second and third row). First, as consistently investigated in many prior research (Snell et al., 2024), both the distilled model and large models showed better performance in general. For example, either POPO R1-Distill-1.5B (59.92) and POPO Qwen-Math-7B (63.12) outperformed POPO Qwen-Math-1.5B (53.06). Second, across all base models, we observed that POPO almost performed better on the hard tasks, e.g., AIME 25 36.67 and Olympiad 57.27 using Qwen-Math-7B. Considering the fact that AIME25 was released after the model release, it rejected the potential that POPO performed better due to any potential data leakage in the pretraining phase. Meanwhile, the clear trend that POPO started with comparable performance with GRPO in MATH-500, and later achieved better performance in the AIME 25 (AIME 25 17.59% v.s. MATH-500 1.66%), suggested relative 15.93% improvement. This trend illustrated our assumption that the combinatorial vastness made penalizing a few sampled negatives unlikely to cover a meaningful reward signal under sparse binary rewards. We provided examples of RL responses in Appendix E.

To conclude, the above results provide both model-agnostic and dataset-agnostic evidence for applying POPO to improve mathematical reasoning ability.

Table 2. Ablation study on importance sampling and weight redistribution. The default setup is without (w/o) negative rollouts and with weight redistribution. w and w/o stand for with and without.

	AIME 2025	**Olympiad**	**Avg.**
Default	23.33	51.19	37.26
w/o negative ; w uniform	16.67	49.11	32.89
w negative ; w/o uniform	13.33	48.07	30.70
w negative ; w uniform	6.67	39.32	23.00

3.3. Ablation and Sweeping Studies

Having established the results in Table 1, we next investigated several key components in our POPO, i.e., importance sampling, weight redistribution, and momentum adaptation, and representation-space alignment via ablation and sweeping studies. We took Qwen-Math-1.5B as our base model.

Importance sampling and weight redistribution. We first considered the situation where we removed the positive rollout constraints, indicating the same rollout usage as GRPO. We surprisingly found that POPO still maintained reasonable performance (30.70 v.s. default 37.26) across AIME 25 and Olympiad benchmarks, which contradicted our intuition. Recall that in the GRPO, we have grouped relative advantage estimation to penalize the negative rollouts. In contrast, POPO respects all rollouts for positive reinforcement. We assumed this magic was due to the weight redistribution, which guaranteed the negative gradient, by implicitly penalizing the negative rollouts. In the ablation, we removed the redistribution weight by treating all rollouts uniformly. In Table 2, we observed that POPO's performance dropped drastically (23.00). **Momentum adaptation and alignment.** In Table 3, we observed that by removing the momentum adaptation, the POPO performances dropped to 30.85, indicating the necessity of slowing the adaptation of the anchor model. This observation also aligned with recent studies that a static reference model may limit the

post-training RL performance. Table 3 last row showed that without the representation-space alignment, the policy training was unstable, causing the performance drop. **Sweep study.** We presented studies for predictor layers, predictor dimensions, and momentum rate in the Appendix C.

Table 3. Ablation study on momentum adaptation and representation-space alignment. The default setup has both momentum and alignment components.

	AIME 2025	Olympiad	Avg.
Default	23.33	51.19	37.26
w/o momentum	13.33	48.37	30.85
w/o alignment	13.33	50.15	31.74

4. Discussion and Conclusion

We propose the POPO framework that enables effective post-training RL via only positive rollouts. In recent RLVR research, we also note that POPO shares a similar spirit with recent works. For example, (Cui et al., 2026a) uses a contrastive head to align reasoning paths within the semantic space. However, POPO is unique in combining innovative components, including bounded importance sampling, adaptive EMA anchoring, and representation-space alignment. Our results illustrate the potential implications of POPO in future RLVR. However, our work has a few limitations. First, we design POPO for sparse binary rewards, yet many practical scenarios are directly involved with dense reward signals (Setlur et al., 2024; Cui et al., 2025). We will extend our POPO by introducing an adaptive rollout partition mechanism for better utilizing negative rollouts. Second, POPO is validated only on text-only LLMs with purely mathematical reasoning benchmarks. We will extend POPO to other domains such as code generation tasks (Le et al., 2022) and multimodal reasoning (Lu et al., 2023), and generative modeling (Cui et al., 2026b).

Disclose Use of LLMs

We used LLMs, such as Claude and GPT, for initial idea brainstorming and later to check grammar and spelling and improve the clarity of our manuscript's language. All scientific contents, including figures, tables, experimental design, analysis, and interpretations, are entirely the original work of the authors.

Impact Statement

All authors of this study strictly adhere to the ICML ethical guidelines. Our research does not involve any potential conflicts of interest or sponsorship issues. We have carefully considered and addressed issues related to discrimination, bias, and fairness in our methodology. This study does not involve any privacy or security concerns, fully complies with legal requirements, and upholds the highest standards of research integrity. All experimental procedures and data processing follow established ethical guidelines for machine learning research.

References

Agarwal, R., Vieillard, N., Zhou, Y., Stanczyk, P., Ramos Garea, S., Geist, M., and Bachem, O. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Balunović, M. et al. Matharena: Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.

Chen, M. et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, X. and He, K. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

Cobbe, K. et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Cui, G. et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Cui, S. et al. Clipo: Contrastive learning in policy optimization generalizes rlvr. *arXiv preprint arXiv:2603.10101*, 2026a.

Cui, Z., Liu, X., Fang, H., Xu, M., Liu, J., Xu, Z., Pian, W., Deng, S., Du, F., Ge, C., et al. Do joint audio-video generation models understand physics? *arXiv preprint arXiv:2605.07061*, 2026b.

Grill, J.-B. et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

Gu, Y., Dong, L., Wei, F., and Huang, M. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Guo, D. et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

He, C. et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 3828–3850, 2024.

Hendrycks, D. et al. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jaech, A. et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Kwon, W. et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Lambert, N. et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Le, H. et al. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 21314–21328, 2022.

Lewkowycz, A. et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Li, J. et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Liu, T. et al. Statistical rejection sampling improves preference optimization. *arXiv preprint arXiv:2309.06657*, 2023.

Lu, P. et al. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Luo, M. et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 3(5), 2025.

Mobahi, H., Farajtabar, M., and Bartlett, P. Self-distillation amplifies regularization in hilbert space. *Advances in Neural Information Processing Systems*, 33:3351–3361, 2020.

Ouyang, L. et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Schulman, J. et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Setlur, A. et al. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Shao, Z. et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Snell, C. et al. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Wang, X. et al. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wang, Y. et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.

Wei, J. et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yang, A. et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Yue, Y. et al. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Appendix

A. Training and Evaluation Benchmarks

We train on DeepScaleR-Preview-Dataset and evaluate on five benchmarks spanning elementary through Olympiad-level difficulty.

- DeepScaleR-Preview-Dataset (Luo et al., 2025): a curated training corpus of approximately 40,000 unique mathematics problem–answer pairs compiled from AIME problems (1984–2023), AMC problems (before 2023), the Omni-MATH dataset, and the Still dataset, spanning difficulty levels from high-school competition to advanced mathematical reasoning.
- MATH-500 (Hendrycks et al., 2021): a 500-problem representative subset of the MATH test set, from the MATH benchmark that OpenAI created in their *Let's Verify Step by Step* (Jaech et al., 2024) paper.
- AMC 2023 (Li et al., 2024): a 40-problem benchmark curated from the 2023 AMC 12A and AMC 12B contests, covering short-answer high-school competition mathematics.
- AIME 2024 (Li et al., 2024): a 30-problem benchmark drawn from the 2024 AIME I and AIME II exams, consisting of challenging high-school competition problems with numerical final answers.
- AIME 2025 (Balunović et al., 2025): a 30-problem benchmark drawn from the 2025 AIME I and AIME II exams; because it was released in 2025, it is commonly used as a fresher benchmark with lower contamination risk for models whose pretraining predates the contest.
- OlympiadBench (He et al., 2024): the English mathematics portion of OlympiadBench, an Olympiad-level bilingual multimodal scientific benchmark; the official release provides English-math subsets spanning text-only and multimodal problems, with both open-ended and proof-based variants.

B. Proof of Theorem 2.1: Implicit Negative Gradient

Proof. We analyze the gradient of $\mathcal{L}_{\text{POPO}}$ with respect to the logit $z_{y'}$ for an incorrect response $y' \in \mathcal{S}^-(x)$. Recall the POPO loss in the main text equation (1):

$$\mathcal{L}_{\text{POPO}}(\theta) = \underbrace{-\sum_{y \in \mathcal{S}^+(x)} w_\theta(y \mid x) \log \pi_\theta(y \mid x)}_{\mathcal{L}_{\text{NLL}}} + \alpha \cdot \mathcal{L}_{\text{sim}}(\theta, \phi, \xi) + \beta \cdot \mathcal{L}_{\text{ent}}(\theta). \tag{1}$$

We compute the gradient contribution of each component separately. Throughout, we work with a single prompt x and drop the conditioning for brevity.

Step 1: Softmax preliminary. Under the softmax parameterization $\pi_\theta(y \mid x) = \exp(z_y)/\sum_{y''} \exp(z_{y''})$, the standard softmax Jacobian gives

$$\frac{\partial \pi_\theta(y \mid x)}{\partial z_{y'}} = \pi_\theta(y' \mid x)\big(\delta_{y,y'} - \pi_\theta(y \mid x)\big), \tag{2}$$

$$\frac{\partial \log \pi_\theta(y \mid x)}{\partial z_{y'}} = \delta_{y,y'} - \pi_\theta(y' \mid x), \tag{3}$$

where $\delta_{y,y'}$ is the Kronecker delta. Since $y \in \mathcal{S}^+(x)$ and $y' \in \mathcal{S}^-(x)$ are in disjoint sets, we have $\delta_{y,y'} = 0$, and thus

$$\frac{\partial \log \pi_\theta(y \mid x)}{\partial z_{y'}} = -\pi_\theta(y' \mid x) \quad \text{for all } y \in \mathcal{S}^+(x). \tag{4}$$

Step 2: Weight invariance for negative logits. The bounded importance weights: $w_\theta(y \mid x) = \pi_\theta(y \mid x)/Z^+(x)$ where $Z^+(x) = \sum_{y'' \in \mathcal{S}^+(x)} \pi_\theta(y'' \mid x)$. We show that these weights are invariant to perturbations of negative logits. For $y \in \mathcal{S}^+(x)$ and $y' \in \mathcal{S}^-(x)$, by the quotient rule:

$$\frac{\partial w_\theta(y \mid x)}{\partial z_{y'}} = \frac{1}{Z^+} \frac{\partial \pi_\theta(y \mid x)}{\partial z_{y'}} - \frac{\pi_\theta(y \mid x)}{(Z^+)^2} \frac{\partial Z^+}{\partial z_{y'}}. \tag{5}$$

From (2) with $y \neq y'$:

$$\frac{\partial \pi_\theta(y \mid x)}{\partial z_{y'}} = -\pi_\theta(y \mid x)\, \pi_\theta(y' \mid x). \tag{6}$$

Summing over the positive set:

$$\frac{\partial Z^+}{\partial z_{y'}} = \sum_{y'' \in \mathcal{S}^+(x)} \frac{\partial \pi_\theta(y'' \mid x)}{\partial z_{y'}} = -\pi_\theta(y' \mid x) \sum_{y'' \in \mathcal{S}^+(x)} \pi_\theta(y'' \mid x) = -\pi_\theta(y' \mid x)\, Z^+. \tag{7}$$

Substituting back into (5):

$$\begin{aligned} \frac{\partial w_\theta(y \mid x)}{\partial z_{y'}} &= \frac{-\pi_\theta(y \mid x)\, \pi_\theta(y' \mid x)}{Z^+} - \frac{\pi_\theta(y \mid x)}{(Z^+)^2} \big(-\pi_\theta(y' \mid x)\, Z^+\big) \\ &= \frac{-\pi_\theta(y \mid x)\, \pi_\theta(y' \mid x)}{Z^+} + \frac{\pi_\theta(y \mid x)\, \pi_\theta(y' \mid x)}{Z^+} \;=\; 0. \end{aligned} \tag{8}$$

This invariance has a clear interpretation: rescaling all probabilities by the same factor (as the softmax does when a single logit changes) leaves the *ratio* $\pi_\theta(y \mid x)/Z^+$ unchanged. Crucially, this means the weight normalization over $\mathcal{S}^+(x)$ does not attenuate the implicit gradient on negatives.

Step 3: NLL gradient on negative logits. Applying the product rule and using (8):

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{NLL}}}{\partial z_{y'}} &= -\sum_{y \in \mathcal{S}^+(x)} \left[\frac{\partial w_\theta(y \mid x)}{\partial z_{y'}} \cdot \log \pi_\theta(y \mid x) + w_\theta(y \mid x) \cdot \frac{\partial \log \pi_\theta(y \mid x)}{\partial z_{y'}} \right] \\ &= -\sum_{y \in \mathcal{S}^+(x)} w_\theta(y \mid x) \cdot \big(-\pi_\theta(y' \mid x)\big) \\ &= \pi_\theta(y' \mid x) \sum_{y \in \mathcal{S}^+(x)} w_\theta(y \mid x) \;=\; \pi_\theta(y' \mid x), \end{aligned} \tag{9}$$

where the last equality uses $\sum_{y \in \mathcal{S}^+(x)} w_\theta(y \mid x) = 1$. This term is unconditionally positive: gradient descent on the NLL loss alone already decreases the probability of every incorrect response.

Step 4: Entropy gradient on negative logits. The entropy loss is $\mathcal{L}_{\text{ent}} = -H(\pi_\theta(\cdot \mid x)) = \sum_y \pi_\theta(y \mid x) \log \pi_\theta(y \mid x)$. Its gradient with respect to $z_{y'}$ is:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{ent}}}{\partial z_{y'}} &= \sum_y \frac{\partial \pi_\theta(y \mid x)}{\partial z_{y'}} \big(1 + \log \pi_\theta(y \mid x)\big) \\ &= \sum_y \pi_\theta(y' \mid x) \big(\delta_{y,y'} - \pi_\theta(y \mid x)\big) \big(1 + \log \pi_\theta(y \mid x)\big) \\ &= \pi_\theta(y' \mid x) \Big[\big(1 + \log \pi_\theta(y' \mid x)\big) - \underbrace{\sum_y \pi_\theta(y \mid x) \big(1 + \log \pi_\theta(y \mid x)\big)}_{= 1 - H(\pi_\theta(\cdot \mid x))} \Big] \\ &= \pi_\theta(y' \mid x) \big[\log \pi_\theta(y' \mid x) + H(\pi_\theta(\cdot \mid x)) \big]. \end{aligned} \tag{10}$$

Step 5: Similarity loss does not contribute at the logit level. The similarity loss in the main text equation (2) depends on the negative logit $z_{y'}$ only through the weights $w_\theta(y \mid x)$. By Step 2, $\partial w_\theta(y \mid x)/\partial z_{y'} = 0$, and the cosine similarity term involves hidden representations $f_\theta(x, y)$ for $y \in \mathcal{S}^+(x)$, which are computed from intermediate layers and do not depend on the output logit $z_{y'}$. Therefore, $\partial \mathcal{L}_{\text{sim}}/\partial z_{y'} = 0$.

Step 6: Combining and sign analysis. Summing the contributions from Steps 3–5:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{POPO}}}{\partial z_{y'}} &= \frac{\partial \mathcal{L}_{\text{NLL}}}{\partial z_{y'}} + \beta \frac{\partial \mathcal{L}_{\text{ent}}}{\partial z_{y'}} + \alpha \frac{\partial \mathcal{L}_{\text{sim}}}{\partial z_{y'}} \\ &= \pi_\theta(y' \mid x) + \beta\, \pi_\theta(y' \mid x)\big[\log \pi_\theta(y' \mid x) + H(\pi_\theta(\cdot \mid x))\big] + 0 \\ &= \pi_\theta(y' \mid x)\Big[1 + \beta\big(\log \pi_\theta(y' \mid x) + H(\pi_\theta(\cdot \mid x))\big)\Big]. \end{aligned} \tag{11}$$

Since $\pi_\theta(y' \mid x) > 0$, the sign is determined by the bracketed factor. Setting it to zero:

$$1 + \beta\big(\log \pi_\theta(y' \mid x) + H\big) = 0 \iff \pi_\theta(y' \mid x) = \exp\Big(-\tfrac{1}{\beta} - H\Big). \tag{12}$$

The gradient is strictly positive whenever $\pi_\theta(y' \mid x)$ exceeds this threshold. In practice, this condition is typically satisfied (as $\exp\Big(-\frac{1}{\beta} - H\Big)$ is close to 0) for any response with non-negligible probability, which completes the proof. □

The proof reveals two reinforcing forces that drive the implicit negative gradient. The first force originates from the weighted NLL loss on positives: through the softmax partition function, reinforcing correct responses imposes a uniform "probability tax" of $\pi_\theta(y' \mid x)$ on every incorrect logit. Notably, the bounded importance weights $w_\theta(y \mid x)$ do not attenuate this force (as shown in Step 2, $\partial w_\theta(y \mid x)/\partial z_{y'} = 0$ for all $y' \in \mathcal{S}^-(x)$), so the weight normalization over $\mathcal{S}^+(x)$ passes through without affecting the implicit gradient on negatives. The second force arises from the entropy regularization equation (10), which adds the strongest suppression (as the penalty) on the most probable incorrect responses.

Algorithm 1 Positive-Only Policy Optimization (POPO)

Require: Dataset $\mathcal{D}$, initial parameters θ_0, learning rate η, EMA momentum τ,
similarity weight α, entropy weight β, group size G

```
Initialize Siamese policy network parameters: ξ ← θ_0
for iteration t = 1, 2, ..., T do
    Sample batch of math question as prefix prompts {x_1, ..., x_B} from D
    for each prompt x_i in batch do
        Generate G responses: {y_1, ..., y_G} ~ π_θ(· | x_i)
        Verify correctness: R_j = R(x_i, y_j) for j = 1, ..., G
        Identify positive set: S^+(x_i) = {y_j : R_j = 1}
    end for
    Collect positive pairs: B^+ = {(x_i, y) : y ∈ S^+(x_i)}
    if B^+ is non-empty then
        for each prompt x with positives do
            Compute weights w_θ(y | x) via w_θ(y | x) = π_θ(y | x)/Z^+(x)
        end for
        Compute POPO loss via Equation (1)
    end if
    Update via EMA: ξ ← τξ + (1 − τ)θ
end forTrained policy network π_θ
```

C. Sweep Studies

We presented the sweep studies of major hyperparameters in Appendix Table 1.

D. Training Details

Infrastructure. All experiments are conducted on a single Kubernetes (K8s) node equipped with 4× NVIDIA A100 80GB GPUs. We use PyTorch with Fully Sharded Data Parallelism (FSDP) for distributed training and vLLM (server mode) for rollout generation. It takes around 11 hours to run POPO for 1 epoch training on 1.5B models and 24 hours on 7B models, respectively. It takes 15 minutes for vLLM inferences.

Table 1. Sweep study on predictor layer, embedding dimension, and momentum rate.

Parameter	**Avg.**
Predictor layers = 2	53.06
Predictor layers = 3	52.45
Predictor layers = 4	51.97
Predictor dimension = 1024	52.38
Predictor dimension = 2048	52.64
Predictor dimension = 4096	53.06
Momentum = 0.99	52.13
Momentum = 0.999	53.06
Momentum = 0.9999	52.21

Hyperparameters. For GRPO, we use clipping parameter $\epsilon = 0.2$ and KL coefficient $\beta_{\text{KL}} = 0.01$.

Table 2. Shared training hyperparameters across RL methods.

Hyperparameter	**Value**
Optimizer	AdamW
Learning rate	1e-7
LR schedule	Cosine decay
Epoch	1
Per-device batch size	32
Gradient accumulation steps	4
Group size	8
Max prompt length	512
Max completion length	1024
Generation temperature	1.0
Weight decay	0.01
Gradient clipping	1.0
Num iterations	1
Precision	bfloat16

Table 3. POPO hyperparameter configuration.

Hyperparameter	**Value**
EMA momentum	0.999
Similarity weight	0.1
Entropy coefficient	0.01
Feature noise std	0.02
Reward threshold	0.0
Predictor hidden dimension	4096
Predictor layers	2
Sampling temperature	0.7
Top-p	0.95
Pass k	8
Samples n	128
Max output length (evaluation)	4096
Inference engine	vLLM
Mode	server
Tensor parallel size	1

E. Examples Responses from POPO and Other RL Algorithms

E.1. Qwen2.5-Math-7B Model on Olympiad Dataset

Input prompt:
$< |im\ start| >$ **System.**
Please reason step by step, and put your final answer within boxed $\{\}$ $< |im\ end| >$**.**
$< |im\ start| >$ **User.**
"Question" $< |im\ end| >$**.**
$< |im\ start| >$ **Assistant**

Question: For a positive integer a, define a sequence of integers $x_1, x_2, \ldots$ by letting $x_1 = a$ and $x_{n+1} = 2x_n + 1$ for $n \geq 1$. Let $y_n = 2^{x_n} - 1$. Determine the largest possible k such that, for some positive integer a, the numbers $y_1, \ldots, y_k$ are all prime.
Ground-truth Answer: 2

POPO Response: To determine the largest possible k such that, for some positive integer a, the numbers $y_1, y_2, \ldots, y_k$ are all prime, we start by analyzing the given sequence and the properties of the numbers involved. Given: - $x_1 = a$ (where a is a positive integer) - $x_{n+1} = 2x_n + 1$ for $n \geq 1$ - $y_n = 2^{x_n} - 1$ We need to find the largest k such that all $y_1, y_2, \ldots, y_k$ are prime numbers. First, let's compute the first few terms of the sequence $\{x_n\}$ and corresponding y_n: 1. **For** $n = 1$**:** - $x_1 = a$ - $y_1 = 2^a - 1$ - y_1 must be prime. This implies that a itself must be prime since $2^a - 1$ is prime only if a is prime (otherwise, $2^a - 1$ would be composite). 2. **For** $n = 2$**:** - $x_2 = 2x_1 + 1 = 2a + 1$ - $y_2 = 2^{2a+1} - 1$ - y_2 must also be prime. 3. **For** $n = 3$**:** - $x_3 = 2x_2 + 1 = 2(2a + 1) + 1 = 4a + 3$ - $y_3 = 2^{4a+3} - 1$ - y_3 must be prime. 4. **For** $n = 4$**:** - $x_4 = 2x_3 + 1 = 2(4a + 3) + 1 = 8a + 7$ - $y_4 = 2^{8a+7} - 1$ - y_4 must be prime. Continuing this pattern, each subsequent y_n becomes exponentially larger. The challenge is to find an a such that all y_1 through y_k are prime. To maximize k, we should start with the smallest possible prime a, which is $a = 2$: - $y_1 = 2^2 - 1 = 3$ (prime) - $y_2 = 2^5 - 1 = 31$ (prime) - $y_3 = 2^{11} - 1 = 2047$ (not prime, since $2047 = 23 \times 89$) Thus, with $a = 2$, we only get $k = 2$. Trying $a = 3$:
- $y_1 = 2^3 - 1 = 7$ (prime) - $y_2 = 2^7 - 1 = 127$ (prime) - $y_3 = 2^{15} - 1 = 32767$ (not prime, since $32767 = 7 \times 31 \times 151$) Again, $k = 2$. Trying $a = 5$: - $y_1 = 2^5 - 1 = 31$ (prime) - $y_2 = 2^{11} - 1 = 2047$ (not prime) Thus, $k = 1$. It appears that regardless of the choice of a, achieving $k \geq 3$ is not possible because y_3 tends to be composite. Therefore, the maximum value of k for which all y_1 through y_k are prime is $k = 2$. The largest possible k such that $y_1, \ldots, y_k$ are all prime is $\boxed{2}$. Correct

GRPO Response: To solve this problem, we need to analyze the sequence x_n and the corresponding sequence y_n. 1. First, let's define the sequence x_n: - $x_1 = a$ - $x_{n+1} = 2x_n + 1$ 2. From this, we can derive the general form of x_n: - $x_2 = 2x_1 + 1 = 2a + 1$ - $x_3 = 2x_2 + 1 = 2(2a + 1) + 1 = 4a + 3$ - $x_4 = 2x_3 + 1 = 2(4a + 3) + 1 = 8a + 7$ - By induction, we can see that $x_n = 2^{n-1}a + 2^{n-1} - 1 = 2^{n-1}(a + 1) - 1$. 3. Next, we define y_n: - $y_n = 2^{x_n} - 1$ - Substituting x_n into y_n, we get $y_n = 2^{2^{n-1}(a+1)-1} - 1$. 4. To find the largest possible k such that $y_1, y_2, \ldots, y_k$ are all prime, we need to check the primality of the first few terms of y_n. 5. Let's start with y_1: - $y_1 = 2^a - 1$ - For y_1 to be prime, a must be such that $2^a - 1$ is prime. The smallest such a is 2, so $y_1 = 2^2 - 1 = 3$, which is prime. 6. Next, let's check y_2: - $y_2 = 2^{2a+1} - 1$ - For y_2 to be prime, $2a + 1$ must be a prime exponent in the Mersenne prime form. If $a = 2$, then $y_2 = 2^{2 \cdot 2+1} - 1 = 2^5 - 1 = 31$, which is prime. 7. Next, let's check y_3: - $y_3 = 2^{4a+3} - 1$ - For y_3 to be prime, $4a + 3$ must be a prime exponent in the Mersenne prime form. If $a = 2$, then $y_3 = 2^{4 \cdot 2+3} - 1 = 2^{11} - 1 = 2047 = 23 \cdot 89$, which is not prime. So, the largest possible k such that $y_1, y_2, \ldots, y_k$ are all prime is 2. Therefore, the answer is $\boxed{2}$. Correct

SFT Response: To solve the problem, we need to understand the sequence x_n and the sequence y_n. The sequence x_n is defined as follows: $x_1 = a$ and $x_{n+1} = 2x_n + 1$. We can express x_n in terms of a and n. Let's first find a general formula for x_n: $x_2 = 2a + 1$, $x_3 = 2(2a + 1) + 1 = 4a + 3$, $x_4 = 2(4a + 3) + 1 = 8a + 7$. We can observe a pattern here. The general formula for x_n is: $x_n = 2^{n-1}a + (2^{n-1} - 1)$. This can be proven by induction: - Base case: For $n = 1$, $x_1 = a$, which is true. - Inductive step: Assume $x_n = 2^{n-1}a + (2^{n-1} - 1)$. Then, $x_{n+1} = 2x_n + 1 = 2(2^{n-1}a + (2^{n-1} - 1)) + 1 = 2^n a + (2^n - 2) + 1 = 2^n a + (2^n - 1)$. So, the formula holds for x_{n+1}. Now, we need to find the sequence y_n: $y_n = 2^{x_n} - 1 = 2^{2^{n-1}a+(2^{n-1}-1)} - 1$. We need to determine the largest k such that the first k terms of the sequence y_n are all prime. Let's start by checking small values of a and see how many of the first few y_n are prime. For $a = 1$: $x_1 = 1$, $x_2 = 2 \cdot 1 + 1 = 3$, $x_3 = 2 \cdot 3 + 1 = 7$, $x_4 = 2 \cdot 7 + 1 = 15$, $x_5 = 2 \cdot 15 + 1 = 31$. So, $y_1 = 2^1 - 1 = 1$ (not prime), $y_2 = 2^3 - 1 = 7$ (prime), $y_3 = 2^7 - 1 = 127$ (prime), $y_4 = 2^{15} - 1 = 32767$ (not prime), $y_5 = 2^{31} - 1 = 2147483647$ (prime). For $a = 2$: $x_1 = 2$, $x_2 = 2 \cdot 2 + 1 = 5$, $x_3 = 2 \cdot 5 + 1 = 11$, $x_4 = 2 \cdot 11 + 1 = 23$, $x_5 = 2 \cdot 23 + 1 = 47$. So, $y_1 = 2^2 - 1 = 3$ (prime), $y_2 = 2^5 - 1 = 31$ (prime), $y_3 = 2^{11} - 1 = 2047 = 23 \cdot 89$ (not prime). From these calculations, we can see that the largest k such that the first k terms of the sequence y_n are all prime is $k = 3$. We can verify that for $a = 1$, the first three terms y_2, y_3, y_4 are not all prime, but the first two terms y_1, y_2 are prime, and the third term y_3 is prime. Therefore, the largest possible k is $\boxed{3}$.Wrong

BASE Response: For each positive integer n, let $f(n)$ be the sum of the digits in the base-four representation of n and let $g(n)$ be the sum of the digits in the base-eight representation of $f(n)$. For example, $f(2020) = f(133210_4) = 10 = 12_8$, and $g(2020) =$ the digit sum of $12_8 = 3$. Let N be the least value of n such that the base-sixteen representation of $g(n)$ cannot be expressed using only the digits 0 through 9. Find the remainder when N is divided by 1000. 1. **Define the sequence x_n:** - Let $x_1 = a$. - For $n \geq 1$, $x_{n+1} = 2x_n + 1$. 2. **Express x_n in terms of a:** - We start with $x_1 = a$. - $x_2 = 2x_1 + 1 = 2a + 1$. - $x_3 = 2x_2 + 1 = 2(2a + 1) + 1 = 4a + 3$. - $x_4 = 2x_3 + 1 = 2(4a + 3) + 1 = 8a + 7$. - By induction, we can generalize x_n as: $x_n = 2^{n-1}a + (2^{n-1} - 1)$ 3. **Define the sequence y_n:** - $y_n = 2^{x_n} - 1$. 4. **Determine the largest possible k such that $y_1, y_2, \ldots, y_k$ are all prime:** - $y_1 = 2^a - 1$. - $y_2 = 2^{2a+1} - 1$. - $y_3 = 2^{4a+3} - 1$. - $y_4 = 2^{8a+7} - 1$. 5. **Check for primality:** - For y_1 to be prime, a must be a Mersenne prime exponent. - For y_2 to be prime, $2a + 1$ must be a Mersenne prime exponent. - For y_3 to be prime, $4a + 3$ must be a Mersenne prime exponent. - For y_4 to be prime, $8a + 7$ must be a Mersenne prime exponent. 6. **Find the smallest a such that y_1, y_2, y_3, y_4 are all prime:** - The smallest Mersenne prime exponent is 2. - $a = 2$: $y_1 = 2^2 - 1 = 3 \quad$ (prime) $y_2 = 2^{2 \cdot 2+1} - 1 = 2^5 - 1 = 31 \quad$ (prime) $y_3 = 2^{4 \cdot 2+3} - 1 = 2^{11} - 1 = 2047 \quad$ (not prime, $2047 = 23 \times 89$) 7. **Conclusion:** - The largest possible k is 3. $\boxed{3}$ 2. **Define the function $f(n)$:** - $f(n)$ is the sum of the digits in the base-four representation of n. 3. **Define the function $g(n)$:** - $g(n)$ is the sum of the digits in the base-eight representation of $f(n)$. 4. **Find the smallest n such that the base-sixteen representation of $g(n)$ cannot be expressed using only the digits 0 through 9:** - The largest digit in base-sixteen is F (which is 15 in decimal). 5. **Determine the range of $f(n)$:** - The sum of the digits in the base-four representation of n can be at most $3 \times \lfloor \log_4 n \rfloor + 3$. 6. **Determine the range of $g(n)$:** - The sum of the digits in the base-eight representation of $f(n)$ can be at most $7 \times \lfloor \log_8 f(n) \rfloor + 7$. 7. **Find the smallest n such that $g(n) \geq 10$:** - We need to find n such that $g(n) = 10$. 8. **Calculate $g(n) = 10$:** - $f(n)$ must be a number whose digits in base-eight sum to 10. - Possible values for $f(n)$ are 17, 26, 35, 44, 53, 62, 71, 80. 9. **Find the smallest n such that $f(n) = 17$:** - 17 in base-four is 1001_4. - $n = 1001_4 = 1 \cdot 4^3 + 0 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 = 65$. 10. **Verify $g(65) = 10$:** - $f(65) = 1 + 0 + 0 + 1 = 2$. - $g(65) = 2$. 11. **Find the next smallest n such that $f(n) = 17$:** - 17 in base-four is 1001_4. - $n = 1001_4 = 1 \cdot 4^3 + 0 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 = 65$. 12. **Find the next smallest n such that $f(n) = 17$:** - 17 in base-four is 1001_4. - $n = 1001_4 = 1 \cdot 4^3 + 0 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 = 65$. 13. **Conclusion:** - The smallest n such that $g(n) = 10$ is $1001_4 = 65$. $\boxed{1001}$ 14. **Find the remainder when N is divided by 1000:** - $N = 1001$. - $1001 \mod 1000 = 1$. $\boxed{1}$Wrong

E.2. Qwen-Math-1.5B Model on the MATH-500 Dataset

Input prompt:
$<|im\ start|>$ **System.**
Please reason step by step, and put your final answer within boxed $\{\}$ $<|im\ end|>$**.**
$<|im\ start|>$ **User.**
"Question" $<|im\ end|>$**.**
$<|im\ start|>$ **Assistant**

Question: How many positive whole-number divisors does 196 have?
Ground-truth Answer: 9

POPO Response: To find how many positive whole-number divisors 196 has, we first factor it into primes. We have: $196 = 14^2 = (2 \times 7)^2 = 2^2 \times 7^2$ Now, use the divisor-counting formula: If a number has prime factorization $p^a q^b$, then the number of positive divisors is: $(a + 1)(b + 1)$ Here, $a = 2$ and $b = 2$, so the number of positive divisors is: $(2 + 1)(2 + 1) = 3 \times 3 = 9$ Therefore, the number of positive whole-number divisors of 196 is $\boxed{9}$. *Correct*