TRAINING MATHEMATICAL PROOF GENERATORS VIA REFERENCE FREE GENERATIVE VERIFIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mathematical reasoning goes beyond finding correct answers to constructing rigorous proofs. Recent efforts have used reinforcement learning with verifier feedback to improve language models' ability to generate such proofs. However, the long reasoning traces required for generation and verification make reinforcement learning in this setting difficult to stabilize. Through ablations on smaller models, we identify three choices that improve training stability: sequence-level rather than token-level loss aggregation, large batches, and adaptive positive-advantage masking. This masking method controls policy entropy by selectively suppressing updates to low-probability tokens with positive advantage. Starting from Nemotron 3 Ultra 550B, we train a proof generator using feedback from a fixed DeepSeek-Math-V2 verifier that receives only the problem and candidate proof. The resulting model surpasses DeepSeek-Math-V2 itself in proof-generation accuracy on downstream evaluation. The gains persist under an independent judge and reference-aware evaluation.

1 INTRODUCTION

Reinforcement learning (RL) has achieved substantial success in post-training large language models (LLMs) to solve mathematical problems with easily verified final answers (Guo et al., 2025a). Proof-based problems, on the other hand, which are central to advanced mathematics, require rigorous verification of an entire argument rather than a check of its final answer. A correct conclusion does not establish that the reasoning is valid, as human evaluations of LLM-written proofs have repeatedly shown (Mahdavi et al., 2025; Guo et al., 2025b; Dekoninck et al., 2025). Extending RL to proof generation therefore requires a reward signal that reliably reflects proof validity, a substantially harder task than checking final answers. An LLM-based generative verifier can provide a scalable proxy for costly human evaluation by reasoning through a candidate proof before assigning a reward score (Zhang et al., 2024). This allows the verifier to spend additional computation on difficult judgments, and such models are increasingly used to supply rewards for mathematical reasoning and writing (Seed et al., 2025; Lu, 2025).

In this paper, we study training with a fixed, *reference-free* generative verifier. The verifier receives only the problem statement and the candidate proof, with no reference solution or problem-specific grading rubric. Our choice of reference-free feedback is motivated by two findings. First, supplying reference proofs does not consistently improve automated grading accuracy (Dekoninck et al., 2025; Mahdavi et al., 2026). Second, Shao et al. (2025) demonstrated that reference-free feedback can train a strong proof generator within a system that alternately improves generation and verification. Here, we keep the verifier fixed to separate improvements in the generator from changes to the reward model. This lets us ask how much a generator can learn from one verifier, and whether the verifier's own ability to generate proofs limits what it can teach.

Difficult proof problems often require long reasoning traces, posing additional challenges for training efficiency and stability. Our experiments use maximum context lengths of 131K–196K tokens, and the verifier can generate up to 40K tokens to evaluate one proof. We therefore use asynchronous training (Piché et al., 2025; Fu et al., 2025), in which proof generation, verification, and policy updates run concurrently. This reduces time spent waiting for slow verifier calls, but updates can use samples produced by earlier versions of the policy. The challenge is to use these delayed samples without destabilizing the updates.

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.

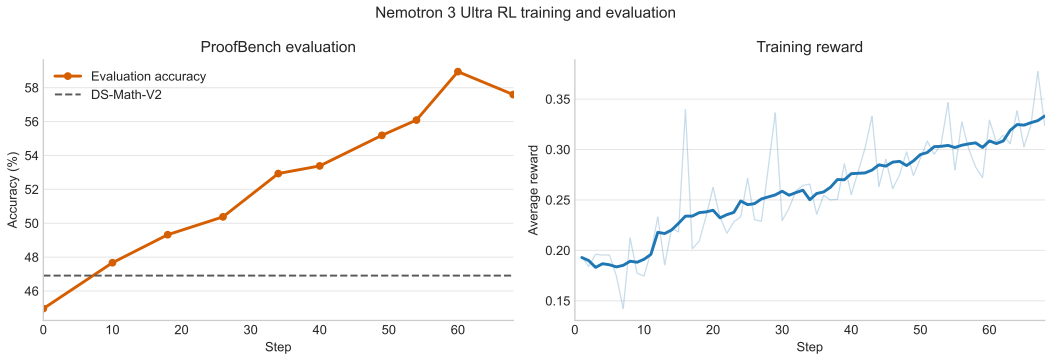

Figure 1: *Left:* The generator's ProofBench25 evaluation accuracy improves with training and significantly exceeds the accuracy of DeepSeek-Math-V2, the verifier model, on the underlying proof-generation task. *Right:* With the carefully tuned reinforcement-learning setup described in Section 4.3, the generator's average training reward increases steadily over the course of training.

To this end, we conduct ablations on smaller-scale models, Qwen3-4B-Thinking and Nemotron 3 Nano 30B, and identify three design choices that improve training stability and sustain RL progress. First, sequence-level loss aggregation delays the reward collapse observed with token-level aggregation. One possible explanation is that token-level aggregation gives more weight to long trajectories, whose generating policies may be further behind the learner. Second, increasing the batch size restores reward improvement at 131K tokens, where a batch configuration effective at 64K tokens no longer suffices. Third, we introduce *adaptive positive-advantage masking* (Ada-POS) to control policy entropy. The method adapts a probability threshold in response to the estimated policy entropy and suppresses updates to low-probability tokens with positive advantage. In our ablation, Ada-POS avoids the entropy spike seen without masking and prolongs reward improvement. These changes improve training stability, but do not prevent degradation later in training (Section 4.3).

Combining these design choices, we train a proof generator starting from Nemotron 3 Ultra 550B using a fixed DeepSeek-Math-V2 verifier. The resulting model surpasses DeepSeek-Math-V2 itself in proof-generation accuracy on downstream evaluation (Figure 1). The gains persist when we use a different grading prompt, supply reference proofs, and replace the training judge with GPT-5.5 (Section 4.4). Thus, a verifier's ability to generate proofs need not be a ceiling on the performance of a generator trained from its judgments.

2 Related Work

Proof-Based Mathematical Reasoning. The success of reinforcement learning with verifiable rewards in final-answer mathematics (Guo et al., 2025a) has spurred interest in proof-based mathematical reasoning, where verifiers must assess the correctness of an entire argument rather than its final answer. Early efforts established benchmarks for measuring LLMs' proof-verification abilities, including the Open Proof Corpus (Dekoninck et al., 2025), ProofBench (Ma et al., 2025), and Hard2Verify (Pandit et al., 2025). More recent work has focused on improving proof generation, with DeepSeekMath-V2 and QED-Nano being the two closest works to ours. DeepSeekMath-V2 (Shao et al., 2025) trains a reference-free verifier to score proofs and uses it as the reward model for a proof generator, improving the two in alternation. Its report emphasizes the final model and does not separately report the starting point of the RL stage or the gains attributable to RL. It also omits several Group Relative Policy Optimization (GRPO) (Shao et al., 2024) configuration details that we find decisive at long context. We study this reward signal with a fixed verifier to isolate the generator's gains and characterize the training choices needed at long context. In this setting, we find that the generator can surpass its verifier's own proof-generation accuracy. QED-Nano (Qu et al., 2026) distills DeepSeekMath-V2 proofs into a 4B model and then applies GRPO with rubric-based rewards. We likewise initialize small generators with DeepSeekMath-V2 distillation, but use reference-free rewards from a verifier that sees only the problem and candidate proof, without a per-problem grading rubric. We also study longer contexts of 131K tokens, where we find that the RL recipe must change

qualitatively. Concurrent work, MaxProof (Chen et al., 2026), also improves proof generation through rubric-based generative-verifier RL.

Reinforcement Learning with Generative Verifiers. Verifiers for mathematical reasoning have evolved from discriminative process reward models (Lightman et al., 2023; Zheng et al., 2025c) to generative verifiers that cast reward modeling as next-token prediction, allowing the judge to reason in a chain of thought and to benefit from additional test-time compute (Zhang et al., 2024; Zheng et al., 2023). Subsequent work trains such judges with reinforcement learning (Whitehouse et al., 2025; Shi & Jin, 2025) and applies them to best-of-N selection (Toshniwal et al., 2025; Liu et al., 2025b); for natural-language proofs in particular, carefully prompted LLM judges approach human grading agreement (Dekoninck et al., 2025; Luong et al., 2025; Ma et al., 2025). Generative verifiers are also increasingly used as the reward signal inside RL itself. Seed1.5-Thinking (Seed et al., 2025) replaces rule-based answer matching with a trained thinking verifier that checks equivalence against a reference answer, reporting fewer reward hacks and more consistent rewards; Nemotron 3 (Blakeman et al., 2025a;b) likewise uses an LLM judge to verify mathematical answers within its multi-environment RL; and Writing-Zero (Lu, 2025) extends the paradigm to non-verifiable writing tasks with a pairwise generative reward model that uses bootstrapped responses as temporary references. These settings still assume a reference answer or a comparison set. In proof generation, however, a correct proof may follow an entirely different argument from a reference solution. Verification therefore requires assessing the validity of the candidate's reasoning regardless of its alignment with a reference solution. We study reference-free rewards that assess this validity using only the problem statement and the candidate proof.

Stabilizing Reinforcement Learning in LLMs. GRPO (Shao et al., 2024) has become the standard for reasoning RL, and much recent work targets the instabilities that appear as responses grow longer, training-inference mismatch increases, and training becomes more off-policy, or studies how to scale RL compute stably (Khatri et al., 2026). DAPO (Yu et al., 2026) decouples the PPO clipping range (Schulman et al., 2017) and introduces dynamic sampling to filter zero-variance prompts, while asynchronous designs such as PipelineRL (Piché et al., 2025) keep accelerators busy through in-flight weight updates at the cost of training on slightly stale rollouts. Several methods revisit how importance ratios should be handled (Roux et al., 2025). At the token level, CISPO (Chen et al., 2025) clips the importance weight rather than dropping the token, so that rare low-probability "fork" tokens keep receiving gradient; IcePop (Team et al., 2025) masks tokens whose training-to-inference probability ratio falls outside a fixed interval, countering the compounding train–inference mismatch of MoE models; and GSPO (Zheng et al., 2025b) replaces token-level ratios with a length-normalized sequence-level ratio and clips whole responses. Routing replay (Zheng et al., 2025a; Liu et al., 2025a) can also stabilize MoE RL, but we did not evaluate it because our training stack lacked an efficient implementation at the time. Our setting pushes these questions to 131K contexts with rewards that each take up to 40K verifier tokens to compute. We adopt an asynchronous pipeline with token-level truncated importance sampling. We find that a sufficiently large batch is a prerequisite for any reward improvement at all in this regime.

3 BACKGROUND

Reward Function. A generator model π_θ is trained to generate proof $y \sim \pi_\theta(\cdot \mid x)$ for a given problem x. A reward function $R(y, x)$ assigns a scalar score to the proof y on the problem x. The goal is to maximize the expected reward over problems and generated proofs.

Reinforcement Learning Algorithm. GRPO (Shao et al., 2024) uses rewards from a group of sampled proofs to train the generator model. For a problem $x \sim \mathcal{D}$, the rollout workers sample a group of G proofs $\{y_i\}_{i=1}^{G}$ from the generation policy π_{gen} and obtain rewards $R_i = R(y_i, x)$. We use the normalized advantage $\widehat{A}_i = (R_i - \mu_i)/\sigma_i$, where μ_i and σ_i are computed from the other rollouts in the group using leave-one-out estimators. In asynchronous RL, generation and optimization run concurrently, so the policy that generated a token can lag behind the optimizer. Following standard asynchronous RL notation, we denote the generation policy by π_{gen} and the current learner policy by $\pi_{\text{train}} = \pi_\theta$, and define the token-level importance ratio

$$\rho_{i,t} = \frac{\pi_{\text{train}}(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{gen}}(y_{i,t} \mid x, y_{i,<t})}. \tag{1}$$

Here, π_{gen} is the policy version that actually sampled the token and may vary across positions under in-flight weight updates.

Let $\omega(\rho, \widehat{A})$ be a generic importance-weighting rule. We write the asynchronous GRPO update directly as the policy-gradient estimator

$$g_\omega(\theta) = \mathbb{E}_{\substack{x\sim\mathcal{D} \\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{gen}}}} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{T_i}\sum_{t=1}^{T_i} \omega(\rho_{i,t}, \widehat{A}_i)\widehat{A}_i \nabla_\theta \log \pi_{\text{train}}(y_{i,t} \mid x, y_{i,<t}) \right], \tag{2}$$

where ω determines how each off-policy token contributes to the gradient. Two standard choices are PPO-style masking and truncated importance sampling. For PPO-style masking,

$$\omega_{\text{PPO}}(\rho, A) = \begin{cases} 0, & A > 0 \text{ and } \rho > 1 + \epsilon_+, \\ 0, & A < 0 \text{ and } \rho < 1 - \epsilon_-, \\ \rho, & \text{otherwise}, \end{cases} \tag{3}$$

For token-level truncated importance sampling (TIS), $\omega_{\text{TIS}}(\rho, A) = \text{clip}(\rho, c_{\min}, c_{\max})$. The first choice is the derivative of the PPO-style asymmetric clipped objective (Yu et al., 2026; Schulman et al., 2017); the second is truncated importance sampling (Chen et al., 2025). Unlike PPO-style masking, TIS retains out-of-range tokens but clamps their importance weights. When $\pi_{\text{gen}} = \pi_{\text{train}}$, both reduce to unit weighting and Equation (2) recovers the ordinary on-policy GRPO update.

4 Training Generators from Verifier Signals

We follow the same setup as in Shao et al. (2025), where a generator model π_θ is trained to generate proof $y \sim \pi_\theta(y|x)$ for a given problem x. The goal is to attain the highest reward $R_{\text{correctness}}(y, x) \in \{0, 0.5, 1\}$ for the proof y on the problem x. The reward is computed by a verifier model v_ϕ that is trained to judge the correctness of the proof y on the problem x. The reward $R_{\text{correctness}}(y, x)$ is 0 if the proof is incorrect or does not follow the required output format, 1 if the proof is correct, and 0.5 if the proof is partially correct. The original final reward in DeepSeek-Math-V2 uses a weighted sum of correctness reward and a self-assessment reward. The self-assessment component $R_{\text{self-assess}} \in \{0, 0.5, 1\}$ assigns a reward based on how accurately the agent self-assesses the correctness of its generated proof. The purpose of the self-assessment component is to encourage the agent to accurately self-assess the correctness of its generated proof to avoid encouraging dishonest behavior. However, in our early experiments, we found that the self-assessment component leads to converging to a lazy agent where the agent gives up on proving the problem and just admits its incorrectness to the verifier model. Incorporating the self-assessment component will likely require tuning the difficulty of the problems to avoid the lazy agent behavior, which we dedicate to future work. See Appendix A for more details on the lazy agent behavior experiments. For the rest of the paper, we will refer to $R_{\text{correctness}}(y, x)$ as the final reward $R(y, x)$.

Figure 1 summarizes the training progress and downstream performance of our Nemotron 3 Ultra 550B generator. On the left, the generator's ProofBench25 accuracy rises above that of DeepSeek-Math-V2, demonstrating that the generator significantly outperforms the generation capabilities of its verifier on the underlying task. On the right, the average training reward increases steadily with the carefully tuned reinforcement-learning setup described in Section 4.3.

4.1 Reinforcement Learning Setup

We use GRPO (Shao et al., 2024) to train the generator model. Our system follows an asynchronous actor–learner design similar to PipelineRL (Piché et al., 2025): proof generation, verifier-based reward computation, and policy optimization proceed concurrently, and the generation workers are refreshed with newer policy weights while requests remain in flight. This design is particularly important in our setting because the verifier may generate up to 40k reasoning tokens before returning a judgment. The resulting evaluation times are long-tailed, so a synchronous implementation would repeatedly leave the training workers idle while waiting for the slowest verifier calls.

The asynchronous pipeline admits a trajectory only if its age, measured as the number of optimizer steps between the policy version that generated it and the policy it is trained with, is at most $a_{\max}$; older trajectories are discarded. For example, a trajectory whose generation or verification takes so long that the policy is updated more than $a_{\max}$ times in the meantime is dropped. As policy lag grows,

the ratios become more variable and training can become unstable, so we select the importance-weighting rule and $a_{\max}$ empirically for our long-sequence setting, using TIS with $a_{\max} = 4$. The TIS bounds are also chosen empirically: we set $c_{\max} = 5$, and $c_{\min} = 0$ because, in our lower-clamp ablation, it avoids the slower learning and lower peak training reward observed with $c_{\min} = 0.2$ (Appendix E).

In addition to token-level importance weighting, we apply sequence-level masking following the Nemotron 3 RL training recipe (Blakeman et al., 2025b) (details in Appendix D). Hyperparameters are listed in Appendix C.

4.2 Experimental Setup

Compute-efficient proxy models. The long reasoning traces required for proof generation make repeated large-scale model training prohibitively expensive: completions in our distillation corpus average approximately 75K tokens, and our training runs must support contexts of up to 131K–196K tokens. To make broad experimental sweeps feasible, we therefore use two smaller backbones: Nemotron 3 Nano 30B (Blakeman et al., 2025b) and Qwen3-4B-Thinking (Yang et al., 2025). Unless otherwise stated, we initialize our reinforcement-learning experiments from checkpoints obtained by supervised distillation of these models, which we call Nemotron 3 Nano 30B-Distill and Qwen3-4B-Thinking-Distill, respectively.

Distillation data. We construct the supervised corpus using DeepSeek-Math-V2 (Shao et al., 2025) as the teacher. For each olympiad-level problem, we sample eight teacher traces with temperature 0.8 and top-$p = 0.95$, allowing up to 100K generated tokens per call. Each trace first contains a proof and the teacher's self-assessment. We then sample three independent verifications of the proof and ask the teacher to meta-verify both its self-assessment and each of the three verifier outputs. After removing malformed records, we convert the resulting conversations into a supervised mixture of proof generation, proof verification, and meta-verification examples. The processed mixture selects at most one proof, one verification, and one meta-verification target from each valid generation, with the meta-verification target alternating between assessments of the self-analysis and of a verifier critique.

Supervised-distillation setup. We train both models in bfloat16 with packed sequences, a global batch size of 512, a micro-batch size of one, and Adam optimization with weight decay 0.01. We use a 10-step learning-rate warmup followed by cosine decay to 10^{-6}; the training jobs are configured for at most 3,000 optimizer steps. For Nemotron 3 Nano 30B, we use a maximum sequence length of 131,072 tokens, a learning rate of 4×10^{-5}, context-parallel size four, and 16 eight-GPU nodes, and initialize RL from the checkpoint at step 400. For Qwen3-4B-Thinking, we use a maximum sequence length of 196,608 tokens, a learning rate of 2×10^{-4}, context-parallel size four, and 16 eight-GPU nodes, and initialize RL from the checkpoint at step 800.

Evaluation Benchmark. We evaluate model checkpoints on *ProofBench25*, a 133-problem natural-language proof-generation suite. It combines 55 problems from 2025 olympiad competitions, 12 problems from the 2025 Putnam Mathematical Competition, the 30 basic and 30 advanced problems from IMO-ProofBench (Luong et al., 2025), and the six problems from IMO 2025.

4.3 Stable RL Training

We now describe the training recipe used to stabilize long-context RL for proof generation. We motivate its main components through experiments that identify failure modes and evaluate the design choices used to address them.

Need for long-context training. For models distilled on long reasoning traces, we find that RL with a 50K-token generation budget primarily shortens generations: training reward improves, but downstream evaluation with a 200K-token budget shows no performance gain. This suggests that reward gains under substantial truncation need not reflect improved long-context reasoning (Appendix B).

Loss aggregation. We first examine how the granularity of loss aggregation affects training stability. The original GRPO objective (Shao et al., 2024; Liu et al., 2025a) aggregates at the sequence level. That is, token losses are averaged within each sequence before being averaged across the group, as in Equation (2). Liu et al. (2025c) instead propose token-level aggregation, which removes the per-sequence normalization by T_i and averages over all tokens in the batch, so that each token

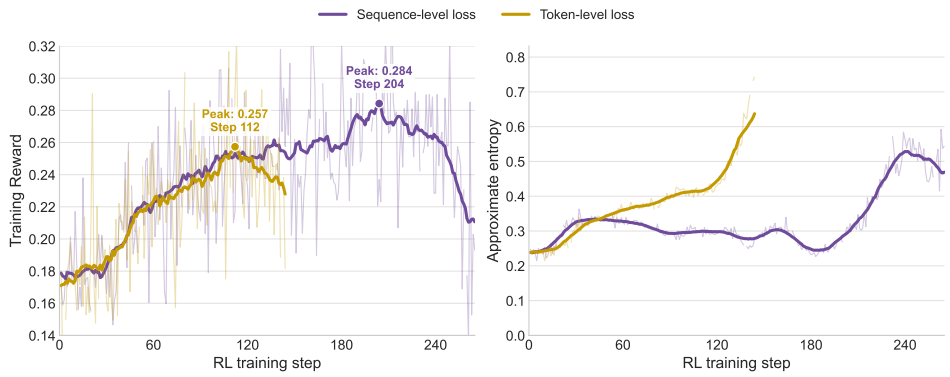

Figure 2: Sequence-level versus token-level loss for a Qwen3-4B-Thinking generator trained with a Qwen3-4B-Thinking-Distill verifier. *Left:* Training reward initially improves under both losses, but the sequence-level loss reaches a higher, later peak before collapsing. *Right:* Approximate entropy rises sharply in both runs, with a later onset under the sequence-level loss. Sequence-level loss delays but does not eliminate training instability.

contributes equally regardless of the length of the sequence to which it belongs. We compare the two variants when training a Qwen3-4B-Thinking generator against a Qwen3-4B-Thinking-Distill verifier (Figure 2). Under both variants, the training reward initially rises but subsequently collapses, and the collapse coincides with a sharp increase in approximate entropy. Sequence-level aggregation, however, delays the onset of this instability. Its reward peaks at 0.284 at step 204, whereas the token-level variant peaks at 0.257 at step 112. Sequence-level aggregation therefore postpones, but does not eliminate, the entropy explosion and ensuing reward collapse.

We conjecture that this difference arises from the coupling between sequence length and off-policyness in asynchronous training. Because longer rollouts take longer to generate, they are more likely to have been sampled from a generator policy that lags further behind the current training policy by the time they reach the learner; in our setting, the most off-policy sequences in a batch are therefore typically also the longest. Since the mismatch between π_{gen} and π_{train} is a well-documented source of instability and collapse in asynchronous RL, the weight assigned to these long sequences directly governs how much off-policy signal enters each update. Consistent with this view, MAI-Thinking(The Microsoft AI Team, 2026) report that long rollouts are more off-policy under asynchronous training and reduce the global learning rate at longer generation lengths to preserve stability. Sequence-level aggregation induces a related effect implicitly: by dividing each sequence's summed token losses by its length T_i, it downweights the per-token contribution of long, and hence typically more off-policy, sequences relative to token-level aggregation, thereby attenuating one driver of collapse. This implicit downweighting offers a plausible account of the delayed instability, although we emphasize that length normalization is only a proxy for, and not an explicit form of, off-policyness-aware weighting.

Adaptive positive-advantage masking for entropy control. The ablation above associates reward collapse with a sharp rise in policy entropy. Qualitatively, we also observe spurious, low-probability tokens appearing in generated responses as training becomes unstable. Because advantages are assigned at the sequence level, these tokens can receive positive reinforcement whenever the overall proof earns an above-average reward. Reinforcing tokens that are unlikely under the current policy can increase entropy, motivating us to mask their positive-advantage updates. We find that this masking reduces approximate entropy relative to the unmasked baseline (Figure 3). To apply this intervention only when needed, we introduce *adaptive positive-advantage masking* (Ada-POS), which adjusts the masking threshold in response to the estimated entropy, without adding an entropy penalty to the objective.

Our adaptive controller is inspired by MAI-Thinking-1 (The Microsoft AI Team, 2026), which uses an importance-weighted entropy estimate and a bounded, fixed-step controller to adjust the upper policy-ratio clipping bound. Their controller tightens the bound when entropy exceeds a target and relaxes it when entropy falls below the target. We adopt this entropy-feedback principle, but instead adapt the minimum-probability threshold for masking positive-advantage tokens, leaving

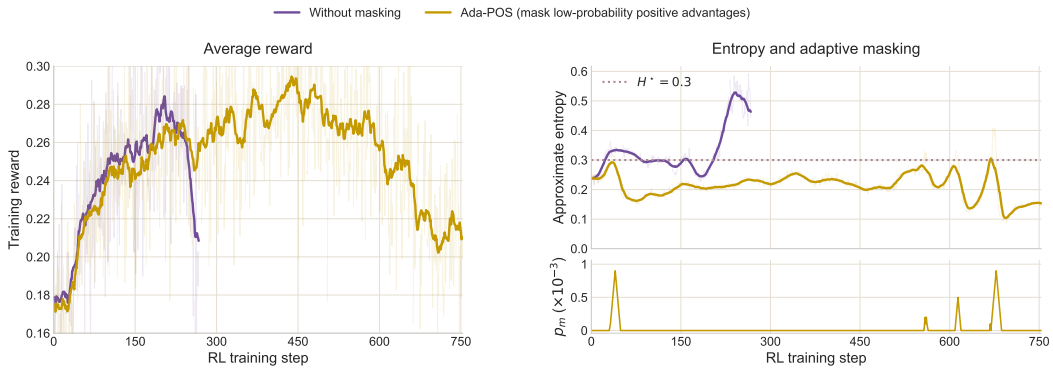

Figure 3: Adaptive positive-advantage masking (Ada-POS) for entropy control. *Left:* Masking low-probability tokens with positive advantages prolongs reward improvement relative to training without masking, although reward still declines later. *Top right:* Approximate entropy with the target $H^\star = 0.3$ indicated by the dotted line. Ada-POS avoids the large entropy spike observed without masking. *Bottom right:* The adaptive probability threshold p_m, shown in units of 10^{-3}, increases intermittently and otherwise returns to zero. This curve reports the masking threshold, not the fraction of tokens masked.

the importance-ratio clipping bounds unchanged. Because their controller acts on the PPO clipping bound ϵ_+, it applies only to ω_{PPO} (Equation (3)); TIS has no such bound to adapt. Ada-POS, in contrast, is a multiplicative mask that is independent of the importance-weighting rule, and can therefore be combined with either ω_{PPO} or ω_{TIS}.

Let k index training steps and let $p_{i,t} = \pi_{\mathrm{train}}(y_{i,t} \mid x, y_{i,<t})$. For a probability threshold $p_m^{(k)}$, we define the token mask $m_{i,t}^{(k)} = 1 - \mathbf{1}\big[\widehat{A}_i > 0 \wedge p_{i,t} < p_m^{(k)}\big]$. We multiply each token contribution in Equation (2) by $m_{i,t}^{(k)}$, treating the mask as fixed during differentiation. Here, "positive" refers to the group-relative advantage, not the absolute reward. The threshold applies to the token's absolute probability under the learner, rather than its importance ratio, and does not modify the rollout sampling distribution. We retain the original loss-normalization denominators, so masked contributions are zeroed rather than redistributed among the remaining tokens.

The threshold is controlled by the approximate entropy measured on the training batch. Writing $\mathcal{V}_k$ for the valid token positions before Ada-POS masking, we use $\widehat{H}_k = -\frac{1}{|\mathcal{V}_k|}\sum_{(i,t)\in\mathcal{V}_k} \rho_{i,t} \log p_{i,t}$. This estimate uses the untruncated training-to-generation importance ratio to correct for off-policy token sampling. Starting from $p_m^{(0)} = 0$, we update the threshold after each training step:

$$p_m^{(k+1)} = \mathrm{clip}\Big(p_m^{(k)} + \delta\ \mathrm{sign}(\widehat{H}_k - H^\star), 0, p_{\max}\Big), \qquad \delta = \frac{p_{\max}}{10}. \tag{4}$$

The controller strengthens masking above the target $H^\star$ and relaxes it below the target, eventually disabling masking when the threshold returns to zero.

Figure 3 compares training with and without Ada-POS. Without masking, approximate entropy rises above 0.5 and reward collapses near step 250. With Ada-POS, entropy remains largely below the target, while reward continues to improve beyond the baseline's collapse, reaching approximately 0.29 near step 450. The threshold is zero for much of training and increases only intermittently, showing that the controller does not require persistent masking. Nevertheless, reward declines later even while entropy remains controlled. Ada-POS therefore mitigates the early entropy-associated instability in this comparison, but entropy control alone does not ensure sustained reward improvement.

Critical Batch Size. Figure 4 reveals a critical batch-size regime for long-context RL. At a maximum sequence length of 64K tokens, we were able to train stably with 32 prompts and 8 generations per prompt. After extending the maximum length to 131K tokens, however, the same 32×8 configuration no longer increases even the training reward. Increasing the batch to 64×16 or 128×16 restores a clear upward trend. Moreover, the 128×16 configuration exhibits a more sustained and stable improvement than 64×16, suggesting that increasing the batch size beyond the critical threshold

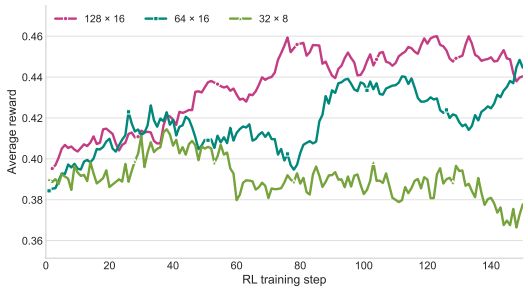

Figure 4: Average training reward at a maximum sequence length of 131K tokens for different RL batch configurations. Each label denotes the number of prompts per step $\times$ the number of generations per prompt. The 32×8 configuration fails to improve, whereas the 64×16 and 128×16 configurations produce increasing reward.

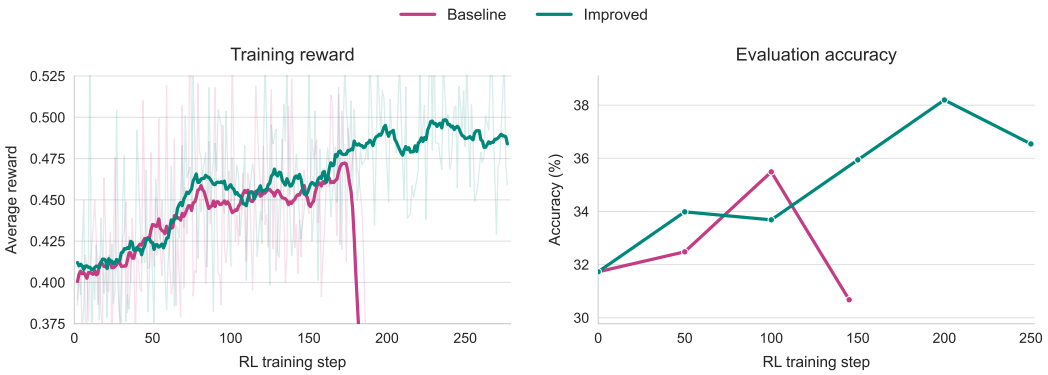

Figure 5: Nemotron 3 Nano 30B-Distill (supervised-distilled Nemotron 3 Nano 30B) under the baseline and improved RL recipes, both using DeepSeek-Math-V2 as the training reward judge. *Left:* The baseline training reward collapses, whereas the improved recipe sustains higher reward over the training horizon shown. *Right:* Evaluation accuracy on ProofBench25, averaging five generated proofs per problem judged by GPT-5.5 with the OPC prompt and a reference proof. Both curves reuse the same initial-checkpoint evaluation. The improved recipe reaches a higher peak accuracy, while the baseline's evaluation degradation is already evident at step 145, more than 20 steps before its visible training-reward collapse.

further reduces optimization noise. Thus, in our setting, longer rollouts require substantially larger batches to achieve a sufficient signal-to-noise ratio for learning.

Combined algorithmic improvements. Figure 5 compares the baseline and improved RL recipes on Nemotron 3 Nano 30B-Distill, our supervised-distilled Nemotron 3 Nano 30B model. The baseline applies the original RL recipe to Nemotron 3 Nano 30B-Distill, with DeepSeek-Math-V2 as the training reward judge. The improved run starts from the same distilled checkpoint and retains the same training judge while applying the combined algorithmic improvements. The plotted evaluation accuracy is measured separately using GPT-5.5 as a judge. The improved recipe avoids the training-reward collapse observed in the baseline over the training horizon shown and achieves higher peak evaluation accuracy: 38.20% at step 200, compared with the baseline's 35.49% at step 100. Thus, the combined improvements yield both more stable training and better downstream performance in this comparison, without implying that evaluation accuracy improves monotonically.

Importantly, evaluation degradation precedes the visible training-reward collapse. Baseline accuracy falls from 35.49% at step 100 to 30.68% at step 145, more than 20 optimization steps before the sharp collapse in the smoothed training-reward curve. Training reward continues to improve in the intervening period, highlighting that it alone is insufficient to detect deteriorating proof-generation performance and motivating periodic checkpoint evaluation with an independent judge.

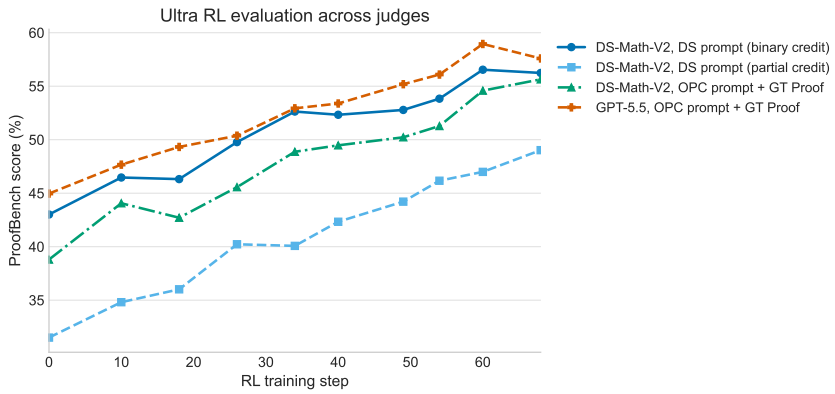

Figure 6: ProofBench25 score throughout RL training under four evaluation protocols. All four scores improve overall, showing that the gains are robust to the judge model, prompt, use of a reference proof, and binary versus partial credit. The DS-Math-V2 partial-credit curve averages scores in $\{0, 0.5, 1\}$, whereas the other curves report binary accuracy, so their scales are not directly comparable.

4.4 Ablation and Analysis

Effect of Different Judges. Figure 6 shows that the model improves overall under all four evaluation protocols. The DeepSeek-Math-V2 (DS) prompt provides the judge with the problem and candidate proof and requests a score in $\{0, 0.5, 1\}$. The partial-credit metric averages these scores directly; this is also the reward used during RL training. Its binary variant instead treats scores of 0.5 and 1 as correct and a score of 0 as incorrect. The other two metrics use the Open Proof Corpus (OPC) evaluation prompt (Dekoninck et al., 2025), which additionally supplies a reference proof when available. We choose this prompt following Mahdavi et al. (2026), who find that it aligns more closely with human judgments than the other prompts they evaluate. We modify the prompt to return a binary correct/incorrect judgment rather than a seven-point score. We evaluate this reference-aware prompt with both DeepSeek-Math-V2 and GPT-5.5. The consistent upward trend indicates that the improvement is not specific to the training judge, or the prompt, or the use of a reference proof.

Proof Length. We also observe that the average proof length increases from 4,699.1 to 7,584.2 characters over the RL run (Appendix F, Figure 10), consistent with the observation in MaxProof (Chen et al., 2026). Although this increase could reflect a form of reward hacking, our qualitative observations primarily indicate more detailed and better-organized proofs.

5 Conclusion

We studied reinforcement learning for proof generation using a fixed, reference-free generative verifier. Training Nemotron 3 Ultra 550B against DeepSeek-Math-V2 produced a generator that surpasses the verifier's own proof-generation accuracy on ProofBench25, and these gains remain consistent under an independent judge, an alternative grading prompt, and reference-aware evaluation. These results indicate that a verifier's ability to generate proofs need not bound the performance of a generator trained on its judgments. To support stable long-context asynchronous training, our proxy-scale ablations identify three effective design choices: sequence-level loss aggregation delays reward collapse, larger batches restore reward improvement at 131K tokens, and adaptive positive-advantage masking (Ada-POS) keeps policy entropy under control.

Limitations and Future Work. Our stability ablations were run on smaller models, Qwen3-4B-Thinking and Nemotron 3 Nano 30B. Verifying these findings at other model sizes and across multiple runs would make them more reliable. Although our design choices allow training to remain stable for much longer, improvements still slow down in very long runs, and a higher training reward does not always lead to higher proof accuracy on benchmarks. Simple indicators that detect this mismatch early would help sustain longer training runs.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods. We have not used generative AI tools for proposing or refining hypotheses, developing conceptual frameworks, formulating mathematical claims, designing or providing feedback on research methodology or experiments, interpreting results, generating synthetic data sets, or qualitative data analysis. Additionally, we used generative AI tools for creating and editing software code, cleaning or reformatting datasets, and for editing the paper to revise paragraphs; we did not use them for brainstorming, discovering research topics, or identifying and summarizing literature. We have reviewed all AI-assisted work: all LLM-generated code was verified and tested for correctness by at least one author. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nvidia nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025a.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025b.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Jiacheng Chen, Xinyu Zhang, Shunkai Zhang, Yanmohan Wang, Lin Li, Tiancheng Qin, Qin Wang, Zhengmao Zhu, Tianle Li, Jingyang Li, et al. Maxproof: Scaling mathematical proof with generative-verifier rl and population-level test-time scaling. *arXiv preprint arXiv:2606.13473*, 2026.

Jasper Dekoninck, Ivo Petrov, Kristian Minchev, Mislav Balunovic, Martin Vechev, Miroslav Marinov, Maria Drencheva, Lyuba Konova, Milen Shumanov, Kaloyan Tsvetkov, et al. The open proof corpus: A large-scale study of llm-generated mathematical proofs. *arXiv preprint arXiv:2506.21621*, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and YI WU. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 36256–36282. Curran Associates, Inc., 2025. doi: 10.52202/085713-1218. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/33c00862bfa29ac72ecf630a41e19352-Paper-Conference.pdf.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Jiaxing Guo, Wenjie Yang, Shengzhong Zhang, Tongshan Xu, Lun Du, Da Zheng, and Zengfeng Huang. Right is not enough: The pitfalls of outcome supervision in training llms for math reasoning. *arXiv preprint arXiv:2506.06877*, 2025b.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Venkata Sai Surya Subramanyam Duvvuri, Manzil Zaheer, Inderjit Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms. In *International Conference on Learning Representations*, volume 2026, pp. 72438–72467, 2026.

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.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Pairjudge rm: Perform best-of-n sampling with knockout tournament. *arXiv preprint arXiv:2501.13007*, 2025b.

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

Xun Lu. Writing-zero: Bridge the gap between non-verifiable problems and verifiable rewards. *arXiv preprint arXiv:2506.00103*, 2025.

Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35406–35430, 2025.

Wenjie Ma, Andrei Cojocaru, Neel Kolhe, Bradley Louie, Robin Said Sharif, Haihan Zhang, Vincent Zhuang, Matei Zaharia, and Sewon Min. Reliable fine-grained evaluation of natural language math proofs. *arXiv preprint arXiv:2510.13888*, 2025.

Hamed Mahdavi, Alireza Hashemi, Majid Daliri, Pegah Mohammadipour, Alireza Farhadi, Samira Malek, Yekta Yazdanifard, Amir Khasahmadi, and Vasant Honavar. Brains vs. bytes: Evaluating llm proficiency in olympiad mathematics. *arXiv preprint arXiv:2504.01995*, 2025.

Sadegh Mahdavi, Branislav Kisacanin, Shubham Toshniwal, Wei Du, Ivan Moshkov, George Armstrong, Renjie Liao, Christos Thrampoulidis, and Igor Gitman. Scaling generative verifiers for natural language mathematical proof verification and selection. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=NsC3NFsavi.

Shrey Pandit, Austin Xu, Xuan-Phi Nguyen, Yifei Ming, Caiming Xiong, and Shafiq Joty. Hard2verify: A step-level verification benchmark for open-ended frontier math. *arXiv preprint arXiv:2510.13744*, 2025.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation. *arXiv preprint arXiv:2509.19128*, 2025.

Yuxiao Qu, Amrith Setlur, Jasper Dekoninck, Edward Beeching, Jia Li, Ian Wu, Lewis Tunstall, Aviral Kumar, et al. Qed-nano: Teaching a tiny model to prove hard theorems. *arXiv preprint arXiv:2604.04898*, 2026.

Nicolas Le Roux, Marc G Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms. *arXiv preprint arXiv:2503.14286*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

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

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Wenlei Shi and Xing Jin. Heimdall: test-time scaling on the generative verification. *arXiv preprint arXiv:2504.10337*, 2025.

Ling Team, Anqi Shen, Baihui Li, Bin Hu, Bin Jing, Cai Chen, Chao Huang, Chao Zhang, Chaokun Yang, Cheng Lin, et al. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *arXiv preprint arXiv:2510.18855*, 2025.

The Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/wp-content/uploads/2026/06/main_20260602_2.pdf.

Shubham Toshniwal, Ivan Sorokin, Aleksander Ficek, Ivan Moshkov, and Igor Gitman. Genselect: A generative approach to best-of-n. *arXiv preprint arXiv:2507.17797*, 2025.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. *arXiv preprint arXiv:2505.10320*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chen Zhang, Chuanqi Tan, Daxin Jiang, Dongdong Zhang, Fandong Meng, Fuli Luo, Guolin Ke, Hang Yan, Hao Tian, Hong Chen, Hongbin Zhang, Hongyi Yuan, Huanbo Luan, Hui Liu, Jiayi Wang, Jidong Zhai, Jindong Chen, Jinhui Liu, Jinqiu Sun, Jiong Cai, Jizhou Huang, Jun Xie, Junyang Lin, Kai Chen, Lei Li, Lei Wang, Liang Wang, Lianxin Jiang, Linjun Shou, Liyang Lu, Lu Chen, Lu Wang, Mengdi Wang, Ming Gong, Ming Yan, Ming Zhou, Nan Duan, Pengcheng Shi, Qian Liu, Qiang Qu, Qianqian Dong, Qiang Wang, Qingsong Wen, Qun Liu, Renjie Zheng, Ruofei Zhang, Rui Wang, Shuming Ma, Shuo Ren, Shuxin Zheng, Songlin Hu, Tao Yu, Tianyu Liu, Tong Xiao, Wei Li, Wei Wang, Wenbin Jiang, Wenhao Huang, Wenxuan Wang, Xiaodong Liu, Xiaofei Sun, Xiaohui Yan, Xiaojun Wan, Xiaolei Wang, Xiaoyan Zhu, Xin Jiang, Xing Xie, Xinyan Xiao, Xipeng Qiu, Xu Tan, Xuefeng Bai, Xuejun Ma, Xun Wang, Yan Gao, Yang Liu, Yanghua Xiao, Yanyan Lan, Yao Meng, Yaqing Wang, Yichong Xu, Yifan Wang, Yihong Chen, Yiming Cui, Yining Wang, Yiqun Liu, Yixuan Su, Yongbin Li, Yongfeng Zhang, Yongxin Lian, Yu Bai, Yu Sun, Yuan Yao, Yuanhang Zhang, Yucheng Wang, Yufan Jiang, Yujie Zhang, Yujing Wang, Yunchang Yang, Yunzhi Yao, Zhenzhong Lan, Zhiwei Zhao, Zhiwu Lu, Zhiye Liu, Zhiwei Wang, Zhiyuan Liu, Zihang Jiang, and Zixuan Zhang. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

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

Chujie Zheng, Kai Dang, Bowen Yu, Mingze Li, Huiqiang Jiang, Junrong Lin, Yuqiong Liu, Hao Lin, Chencan Wu, Feng Hu, et al. Stabilizing reinforcement learning with llms: Formulation and practices. *arXiv preprint arXiv:2512.01374*, 2025a.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025b.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, Vienna, Austria, July 2025c. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

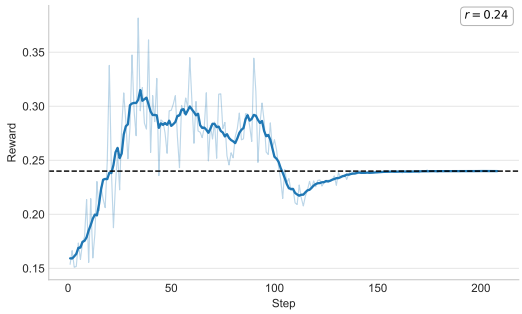

Figure 7: Training reward of the agent when enabling self-assessment reward with $\alpha = 0.76$. The agent converges to a local optima where it always outputs incorrect proofs and self-admits its incorrectness.

A Lazy agent with self-admission reward

Shao et al. (2025) use the following reward to train the proof generator:

$$R(x, y) := \alpha R_{\text{correctness}}(x, y) + (1 - \alpha) R_{\text{self-assess}}(x, y)$$

where $\alpha \in [0, 1]$ is a hyperparameter. Setting $\alpha = 0$ results in the agent only being rewarded for correctness, while setting $\alpha = 1$ results in the agent only being rewarded for self-assessment. Shao et al. (2025) set $\alpha = 0.76$ in their experiments. Here, a local optima is reached where the agent is rewarded for self-admitting its incorrectness to the verifier model, which gives an average reward of 0.24 to the agent. Figure 7 shows the average reward of the agent when setting $\alpha = 0.76$ when training Qwen3-14B (Yang et al., 2025) against a DeepSeek-Math-V2 (Shao et al., 2025) verifier model. As shown, the agent converges to a local optima where it always outputs incorrect proofs and self-admits its incorrectness.

B Need for Long-Context Reinforcement Learning

Models distilled on long reasoning traces can frequently exceed a short RL generation budget, leading to substantial truncation. In our experiments, RL with a maximum generation length of 50K tokens increases average training reward by approximately 0.15 on the 0–1 scale over more than 350 steps, but we observe no downstream performance improvement when evaluation allows up to 200K generated tokens. Instead, the main observed effect is a reduction in generation length. When we restart RL with a 64K-token generation budget, the initial reward is already comparable to the reward reached near the end of the 50K-token run (Figure 8). This comparison is consistent with the shorter-budget run primarily learning to fit its reasoning within the generation limit rather than improving proof-generation ability at a longer evaluation budget. We do not enable the overlong filtering strategy described in DAPO (Yu et al., 2026), which masks the loss of truncated samples, because we observe that enabling it further increases generation length in our setting. These observations motivate long-context RL for models distilled to produce lengthy reasoning traces.

C Hyperparameters

We use Adam with $\beta_1 = \beta_2 = 0.90$. For truncated importance sampling, we set $c_{\min} = 0$ and $c_{\max} = 5$. We set the maximum trajectory age to $a_{\max} = 4$. For sequence-level masking, we set the multiplicative probability-error threshold to 2 (Appendix D). For Ada-POS, we set $H^\star = 0.3$ and $p_{\max} = 10^{-3}$, giving an adjustment of $\delta = 10^{-4}$ per step.

D Sequence-Level Masking

In addition to token-level importance weighting, we multiply each sequence's contribution in Equation (2) by a binary sequence mask

$$m_i^{\text{seq}} = \mathbf{1}\left[\frac{1}{T_i}\sum_{t=1}^{T_i} \exp\left(|\log \rho_{i,t}|\right) \leq 2\right]$$



Figure 8: Average training reward with 50K- and 64K-token generation budgets. The 64K-token run starts at approximately the reward reached by the 50K-token run after more than 350 steps. Despite the training-reward increase in the 50K-token run, we observe no downstream performance gain under a 200K-token evaluation budget. Overlong filtering is disabled in these experiments.

to exclude trajectories with excessive training–generation probability mismatch, following the Nemotron 3 RL training recipe (Blakeman et al., 2025b). This is similar in granularity to the sequence-level clipping in GSPO (Zheng et al., 2025b), but retains our token-level weighting for unmasked sequences. In practice, this mask often affects fewer than 4 sequences in a batch of 2,048 (less than 0.2%).

E TIS Lower-Clamp Ablation

We compare lower importance-weight clamps $c_{\min} = 0$ and $c_{\min} = 0.2$ for token-level truncated importance sampling (ω_{TIS}) when training a Qwen3-4B-Thinking generator with a Qwen3-4B-Thinking-Distill verifier. As shown in Figure 9, setting $c_{\min} = 0$ leads to faster reward improvement and generally higher training reward before collapse. The smoothed reward peaks at 0.284 at step 204 with $c_{\min} = 0$, compared with 0.267 at step 267 with $c_{\min} = 0.2$. Although the positive lower clamp delays the eventual reward collapse, it also slows learning and reduces the peak reward; it does not prevent collapse. We therefore use $c_{\min} = 0$, favoring faster learning and a higher peak reward despite the earlier onset of instability in this ablation.

F Proof Length During Reinforcement Learning

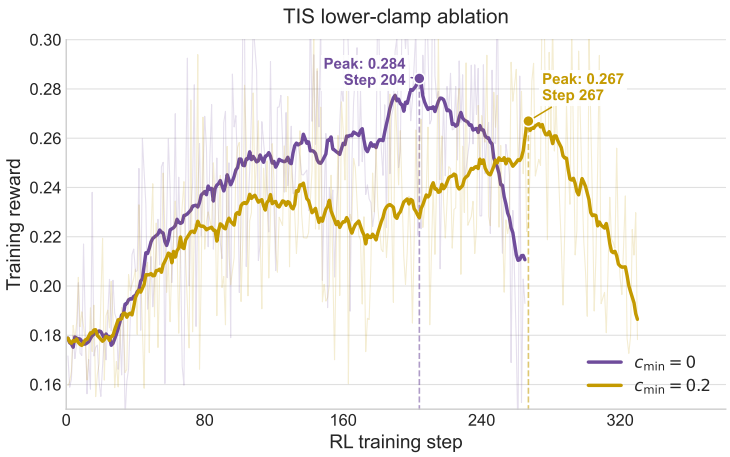

Figure 9: TIS lower-clamp ablation for a Qwen3-4B-Thinking generator trained with a Qwen3-4B-Thinking-Distill verifier. Setting $c_{\min} = 0$ yields faster reward improvement and a higher peak smoothed training reward (0.284 at step 204) than $c_{\min} = 0.2$ (0.267 at step 267), but reward collapses earlier. A positive lower clamp delays rather than prevents collapse.

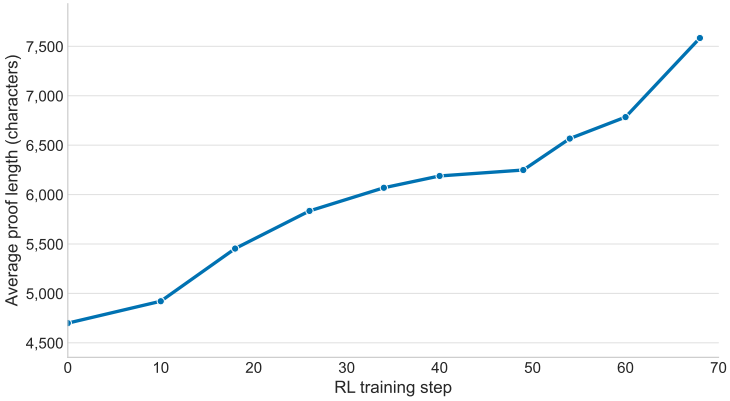

Figure 10: Average proof length in characters over the RL run, increasing from 4,699.1 to 7,584.2 characters.