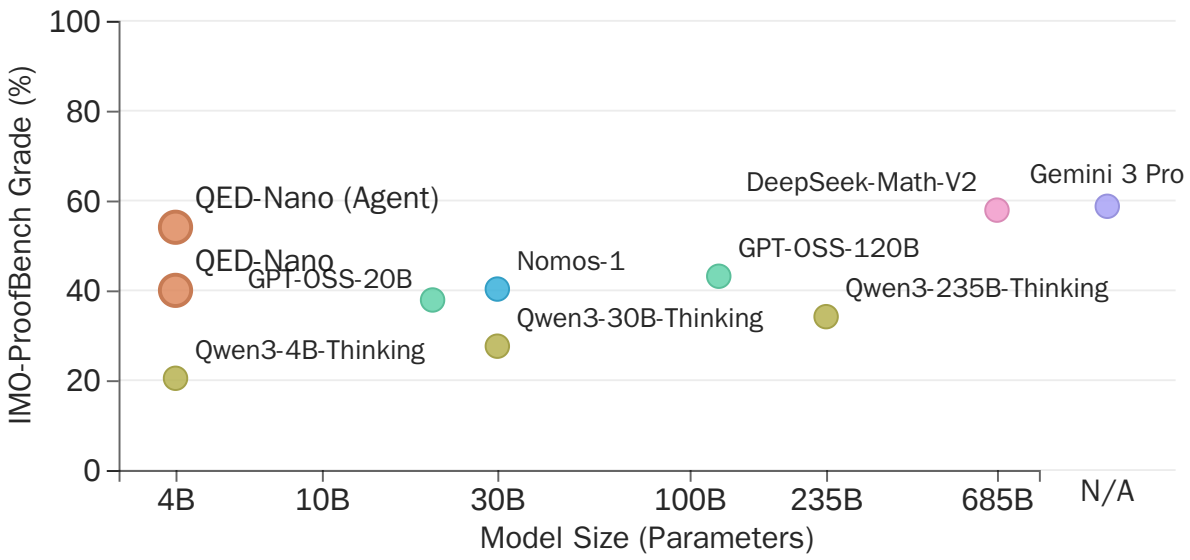
QED-Nano: Teaching a Tiny Model to Prove Hard Theorems



Who Needs a Trillion Parameters? Olympiad Proofs with a 4B Model 🤜

AUTHOR

LM Provers Team[1, 2, 3, 4]

AFFILIATIONS

1. CMU
2. Hugging Face
3. ETH Zurich
4. Numina

PUBLISHED

Feb. 15, 2026

Can we train small language models to solve hard Olympiad-level proof problems at a level close to large frontier models such as Gemini 3 Pro? Yes! We introduce QED-Nano, a compact 4B model post-trained to write Olympiad-level mathematical proofs. Our recipe has three stages: (1) supervised fine-tuning via distillation from DeepSeek-Math-V2, (2) reinforcement learning with dense, rubric-based rewards, and (3) training with a reasoning cache, which decomposes long proofs into iterative summarize-and-refine cycles so the model is capable of continual improvement at test time. Upon deployment, we pair QED-Nano with agentic scaffolds that scale test-time compute to more than 1.5M tokens per problem, combining horizon extension with self-verification. Despite its small size, QED-Nano approaches the proof-writing performance of much larger open and proprietary models at a fraction of the inference cost. We release all models, datasets, grading rubrics, and training code. Concretely, we release:

- The QED-Nano and QED-Nano-SFT models.

- The FineProofs-SFT and FineProofs-RL datasets for post-training our models.
- The training and evaluation code, including the agent scaffolds.

We next describe our approach and results in more detail. Let's dive in!

Introducing QED-Nano: a 4B Model for Olympiad-Level Proofs

Recent proprietary LLM-based systems have demonstrated gold-level performance on the 2025 International Mathematical Olympiad (IMO). However, the training pipelines behind these systems are largely undisclosed, and their reliance on very large models makes them difficult to reproduce or study. This creates a gap between what is possible in principle and what the wider community can realistically build. Our goal is to close this gap between open-source and proprietary systems by showing that small and accessible open models can be trained to attain competitive reasoning performance on these difficult math Olympiad problems.

In this post, we present an end-to-end post-training recipe for building a 4B theorem-proving model. Our model operates entirely in natural language, with no reliance on Lean or external tools. Our recipe is simple and has three components that resemble a typical post-training stack, but with carefully chosen design choices for improving theorem-proving capabilities that we especially tune to scale test-time compute (token budget at test-time):

1. We run supervised fine-tuning (SFT) to imbue the model with a basic ability to write proofs.
2. Then, we perform rubric-based reinforcement learning (RL) with an approach that explicitly optimizes for continual improvement within a long reasoning trace at test time.
3. Finally, we construct test-time scaffolds that allow our model to fully utilize this learned capability of continual improvement in a way that maximizes performance vs tokens spent.

Table 1. Comparison of QED-Nano (4B) with leading open- and closed-source models on IMO-ProofBench, ProofBench, and IMO-AnswerBench. Despite being just 4B in size, QED-Nano matches or exceeds larger models, outperforming Nomos-1 (30B) and Qwen3-235B-A22B-Thinking (50x bigger) on average, while remaining competitive with GPT-OSS-120B. More interestingly, when provided extra test-time compute, QED-Nano (Agent) attains better performance than GPT-OSS-120B on both of the proof-based benchmarks, and it approaches the performance of Gemini 3 Pro, a much stronger proprietary model on IMO-ProofBench.

Model	IMO-ProofBench	ProofBench	IMO-AnswerBench
Qwen3-4B-Thinking-2507	20.4 (2.6)	19.5 (0.9)	55.8
QED-Nano-SFT	39.5 (2.9)	33.3 (0.5)	57.5
QED-Nano	40.0 (0.6)	44.9 (3.4)	67.5
QED-Nano (Agent)	54.0 (3.7)	54.4 (2.4)	-
Qwen3-30B-A3B-Thinking-2507	27.6 (1.0)	26.1 (2.4)	67.0
Qwen3-235B-A22B-Thinking-2507	34.1 (0.7)	33.7 (1.1)	70.5
Nomos-1	40.3 (3.5)	28.3 (3.9)	49.0
GPT-OSS-20B	38.3 (1.2)	38.4 (3.9)	61.5
GPT-OSS-120B	43.1 (3.2)	47.5 (1.7)	70.5
DeepSeek-Math-V2	57.9 (2.0)	60.6 (0.1)	75.8
Gemini 3 Pro	58.7 (2.9)	66.7 (3.1)	83.2

Main Results. Even when just allowed to reason without any scaffold, our trained model, QED-Nano, achieves a 40% score on IMO-ProofBench, 45% on ProofBench, and 68% on IMO-AnswerBench, far better than any other 4B model. On average, these scores make QED-Nano outperform much larger open models such as Nomos-1 (30B) and Qwen3-235B-A22B-Thinking. More importantly, our main result shows that when allowed to reason for up to 1.5 million tokens per problem by pairing the model with a test-time scaffold, QED-Nano (Agent) achieves 54% on IMO-ProofBench and 54% on ProofBench, attaining a strong cost-performance tradeoff on challenging Olympiad-level problems (Figure 2, Table 1). On IMO-ProofBench, this performance is very close to Gemini 3 Pro, a much stronger proprietary model.

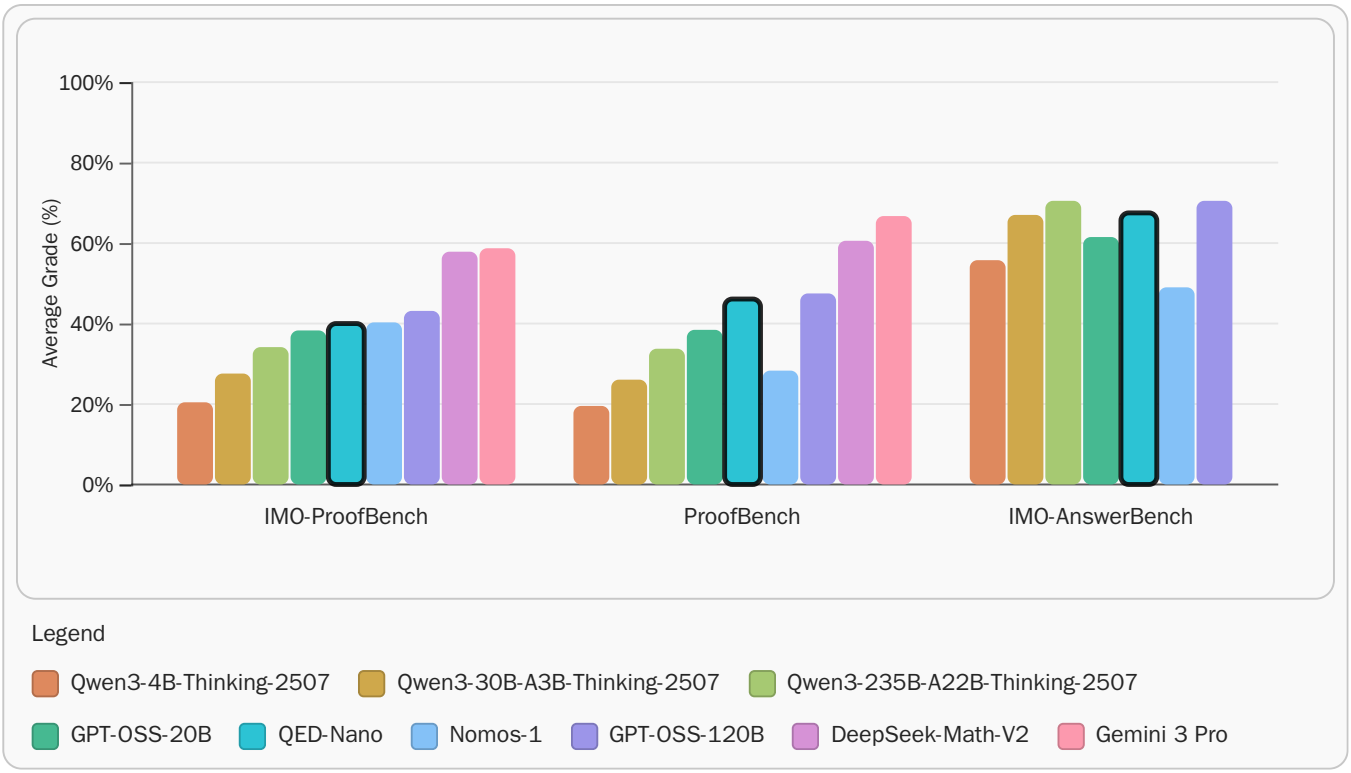


Figure 2. Performance of QED-Nano (4B) within just a single response turn of 50,000 tokens. Even when allowed to reason for just 50,000 tokens (without any form of test-time scaling), QED-Nano roughly matches performance of GPT-OSS-120B and outperforms Nomos-1 on average across the three benchmarks. The only models that considerably outperform QED-Nano are much larger, proprietary models.

Based on estimated inference cost spent via the Hugging Face Hub's inference providers on IMO-ProofBench, QED-Nano (Agent) costs about `$4.0`, and the comparable Gemini 3 Pro run costs `$12.3` under the same accounting. That's ~3× cheaper for similar performance.

Broader implications. Beyond results, we illustrate a broader principle in this blog post: even on the most challenging tasks, we can explicitly train small models to reliably and continually "adapt" at test-time to improve performance. While we showcase our results on Olympiad-style problems (primarily proofs), the recipe we use is generalizable and can also be applied to other domains that allow for rubric-based rewards. More conceptually, in practice, scaling test-time adaptation is often more feasible with smaller models, since inference cost grows quickly with model size. We show that task-specialized small models trained for test-time adaptation can match or exceed much larger generalist systems, suggesting a path toward more capable and specialized models without relying on trillion-parameter architectures that are costly to deploy.

To support further research, we release our SFT dataset, RL prompt set, and an optimized asynchronous and streaming off-policy RL implementation built on pipeline-rl that incorporates our algorithmic improvements for RL with long-horizon reasoning. Our largest RL training run, with rollout length of 50K tokens, fits within 11 nodes of 8xH100s for 4 days, making our approach more accessible and reproducible compared to proprietary approaches. We also discuss early findings, ablations, and small-scale experiments that guided our research workflow and informed

algorithmic and data curation choices, with the goal of helping practitioners apply similar ideas and workflows in their own domains.

Next, we dive into the details of our post-training recipe. In particular, we explain how we source our prompts for RL training and set up an automated proof grading infrastructure. Later, we discuss the two main stages of the post-training recipe (RL, SFT). We discuss details of the SFT training data and RL algorithms that train for our test-time scaffolds in those sections.

Setup: Training Prompts and Grading Schemes

We now discuss how we curate our prompt sets for RL training and design our rubrics, which we use for RL training and evaluation. Training models to generate rigorous Olympiad-level proofs requires carefully curated prompts that are both challenging and clean, with clear criteria for evaluating correctness and mathematical rigor. Therefore, rather than relying on large volumes of loosely curated problem-solution pairs, we construct a compact, high-quality corpus that mirrors the structure and difficulty of competition proofs. Later in this post, we discuss how we reuse this prompt set to collect a dataset for an SFT phase as well. We release all datasets and grading artifacts as standalone resources for the community.

Data source and filtering. We begin with two public datasets: AI-MO/aops, which contains problems sourced from the Art of Problem Solving forums, and AI-MO/olympiads, which aggregates official solutions from a wide range of national and international math competitions (*e.g.*, IMO, USAMO, RMM, *etc.*). While these sources provide coverage, they contain substantial noise, incomplete reasoning, formatting artifacts, and various other issues that preclude them from being seamlessly consumed in any post-training pipeline.

We apply a multi-stage filtering procedure to improve the data quality:

1. We remove problems involving diagrams or images, since our models operate purely in text.
2. We discard trivial or ill-posed entries, including problems where the answer appears directly in the statement, solutions that are implausibly short or purely computational, and materials drawn from easier contests such as AMC or routine exercises. To further enhance solution quality, we run an additional automated filtering pass using GPT-5-Nano. In particular, we prompt it to detect frequent issues observed in the AI-MO/aops dataset, such as questionable problem statements, inconsistencies across proposed solutions, and reference proofs containing substantial logical gaps.
3. Finally, to avoid any contamination with our evaluation benchmarks, we exclude from our training problem set all problems from 2025 competitions and also run a fuzzy string matching

algorithm to weed out any problems similar to those in our evaluation benchmarks. The resulting dataset is a curated collection of Olympiad-style proof problems spanning geometry, number theory, algebra, and combinatorics (see Figure 3).

Next, we discuss how we determine the grading schemes for each problem in this set.

Problem Category Distribution

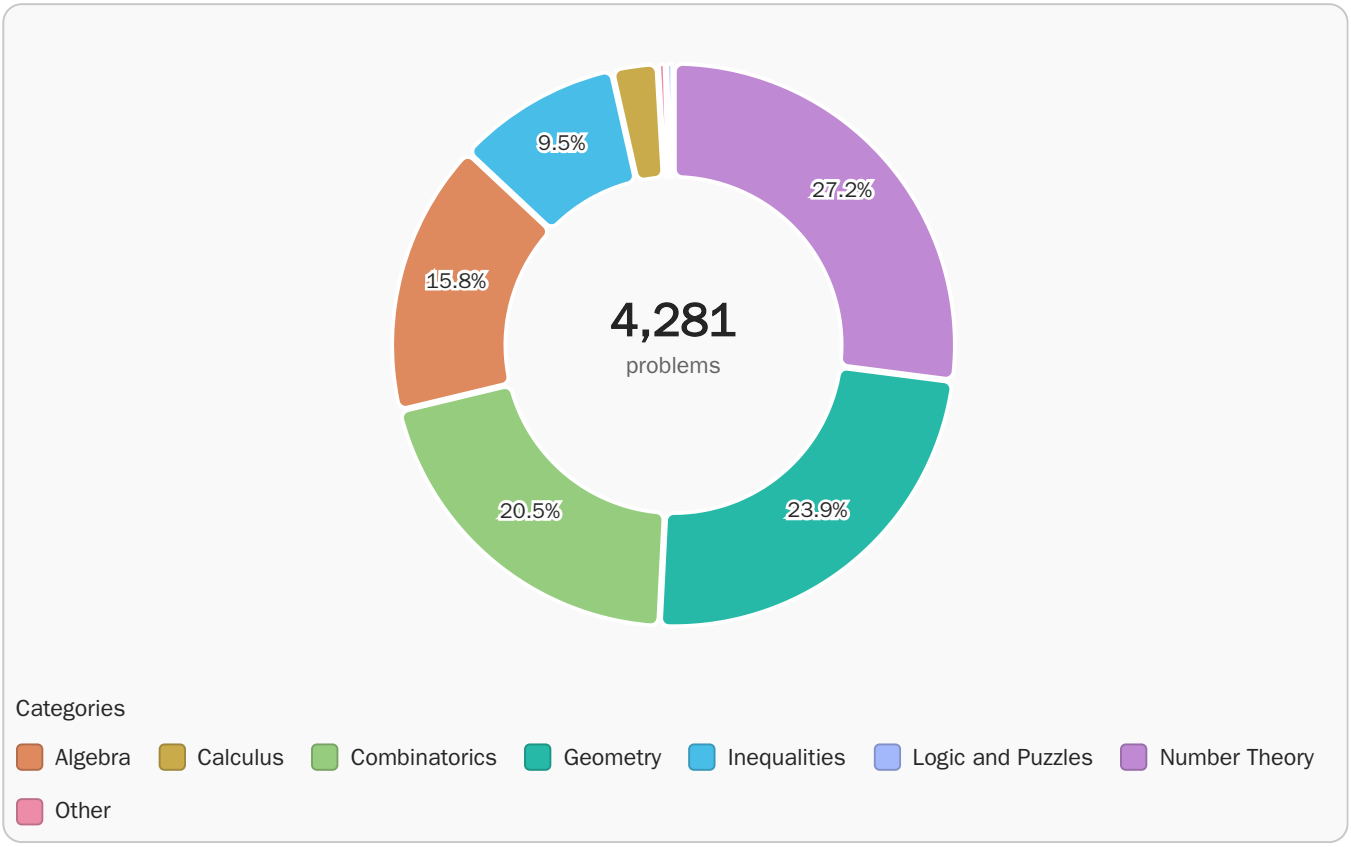


Figure 3. Distribution of 4,281 Olympiad math problems by category. Hover over slices or legend items for detailed counts and percentages. The dataset is dominated by Number Theory (27.2%) and Geometry (23.9%) problems.

Grading schemes. To provide accurate reward signals for training via RL, we construct detailed grading schemes for each problem. Our approach follows the grading framework introduced in ProofBench, which uses Gemini 3 Pro with a custom prompt to generate rubrics that score model solutions from 0 to 7. Each rubric specifies:

1. detailed intermediate checkpoints corresponding to partial correctness
2. common failure modes that warrant zero credit, and
3. specific points where additional deductions are necessary.

As a result, reinforcement learning receives dense, informative feedback instead of sparse success signals, encouraging gradual improvement in long-form reasoning rather than binary outcome optimization. Several examples are shown below.

Example Grading Schemes

Example 1 | Example 2 | Example 3

Let c be fixed natural number. Sequence (a_n) is defined by: $a_1 = 1, a_{n+1} = d(a_n) + c$ for $n = 1, 2, \ldots$. where $d(m)$ is number of divisors of m. Prove that there exist k natural such that sequence $a_k, a_{k+1}, \ldots$ is periodic.

1. Checkpoints (7pts total)

- 1 pt: State or prove the inequality $d(m) \leq \frac{m}{2} + 1$ (or a stronger bound such as $2\sqrt{m}$ for large m) to be used in the boundedness proof.
- 4 pts: Boundedness of the sequence (a_n).
  - 2 pts: Combine the divisor bound with the recurrence to establish an inequality of the form $a_{n+1} \leq \frac{a_n}{2} + C$ (or equivalent logic showing $a_{n+1} < a_n$ for sufficiently large a_n).
  - 2 pts: Conclude that the sequence is bounded (either globally bounded by a value like $2c + 1$ using induction/contradiction, or eventually bounded via infinite descent).
- 2 pts: Periodicity.
  - 1 pt: Apply the Pigeonhole Principle to show that a value in the sequence must repeat.
  - 1 pt: Conclude that repetition implies periodicity because the recurrence relation $a_{n+1} = d(a_n) + c$ is deterministic.

Figure 4. Scoring rubrics used by the evaluation setup.

Problem difficulty annotations. We annotate each problem with a difficulty estimate as determined by the average performance of our base model (Qwen3-4B-Thinking), computed over 128 parallel attempts, graded by GPT-OSS-20B, and using the grading schemes mentioned above. We use these annotations to develop a difficulty-based learning curriculum during RL training. We use this cleaned-up dataset as our main prompt set and release it for others to use.

Our Post-Training Recipe

To develop an effective post-training recipe, we begin by asking a simple question: what does it take for small models to approach the performance of much larger LLMs? At a high level, we achieve this via a reinforcement learning (RL) post-training recipe that trains models to produce long chains-of-thought for proof generation. We therefore first describe our core RL setup, which combines an efficient asynchronous off-policy implementation (that we also release) with rubric-based grading to provide reward signals for policy learning.

While standard RL training should improve the model's proof-writing capability, as we also observe in our experiments, matching the performance of larger models naturally requires small models to use substantially more test-time compute. In our best configurations, this amounts to spending over a million tokens per problem on average. A naive approach that trains RL directly on such long chains of thought is challenging both infrastructure-wise and from the perspective of variance control in long-horizon updates. Instead, we train at moderate output lengths while explicitly optimizing for behavior that benefits from much larger test-time budgets.

To achieve this, we modify our RL recipe to incorporate an algorithmic extension based on the recently introduced Reasoning Cache (RC) approach. During training, the model uses an iterative decoding process that alternates between summarizing its reasoning and continuing to reason conditioned on the generated summary. Incorporating this into training and optimizing rewards under this scaffold allows us to optimize behavior that transfers to other test-time scaffolds used during deployment.

After establishing this post-training recipe on top of the Qwen3-4B base model, we apply the same framework to a stronger initialization that is able to write proofs of higher quality, obtained through offline distillation via supervised fine-tuning. Specifically, we use DeepSeek-Math-V2 (685B parameters) to generate a compact, high-quality set of proof-style examples for supervised mid-training before running RL with RC. We describe this workflow and the associated design decisions, supported by preliminary ablations, in the sections below.

Core Reinforcement Learning Approach

Any typical RL pipeline needs a few basic components: the reward function, the prompt set, and the maximum response length allowed. Along with these components, there are several design questions: How do we decide what length to run RL with? What prompt sets should we use for RL? How do we decide what the grader sees and what rubrics it uses for grading? In this section, we present answers to these questions with some preliminary experiments.

Grading Protocol

Designing a reliable reward signal for RL requires a careful balance between fidelity to human judgment and computational efficiency. A strong grader should produce scores that align closely with human evaluations, while maintaining low latency so that it remains practical for large-scale RL training. To identify an effective configuration, we conducted a series of experiments examining grader model choice, system instructions, and reasoning budget. We evaluate these design decisions below.

Grader evaluation benchmarks. We construct two benchmarks to evaluate our grader design. First, we aggregate all human annotations from the proof-based portion of MathArena, comprising 438 solutions across 22 problems. Second, to obtain a benchmark more representative of our training-time prompt distribution that we will query the grader on, we randomly sample 60 problems from our training corpus. For each problem, we generate four candidate solutions from our base 4B model and the 30B Thinking model from the same model family. We grade these solutions using Gemini 3 Pro, instructed with a prompt adapted from the ProofBench paper, which we found to yield evaluations consistent with human judgment. We therefore treat Gemini 3 Pro's grades as the ground-truth reference in this benchmark. Both of these grader evaluation benchmarks can be found in our Hugging Face collection.

Grader evaluation metric. Both benchmarks contain multiple solutions per problem, enabling calibrated comparisons through a problem-normalized *advantage score*. For each problem p_i and solution y_j^i to problem p_i, we compute the unnormalized advantage $A_{i,j} = r_{i,j} - \overline{r_i}$, where $r_{i,j}$ is the grader-assigned reward to solution y_j^i, and $\overline{r_i}$ is the mean reward across all solutions to problem p_i. Grader accuracy is measured as the mean absolute difference between the candidate grader's advantages and the reference advantages. This formulation removes sensitivity to constant or benign shifts between graders, which is important because such shifts do not affect RL training with several parallel rollouts (as used by GRPO).

Grader model and prompt. Using the metric above, we evaluate five grader prompts drawn from prior work emphasizing different evaluation ideologies (Table 2). On the MathArena subset, GPT-OSS-20B with medium reasoning performs best when paired with the strict ProofBench prompt, which emphasizes strict adherence to the rubric and rejects solutions that deviate from it. Prompts are shown below.

Table 2. Results on the MathArena grading benchmark. Lower is better.

Model	Simple	OPC	ProofBench	ProofBench Strict	GIMO
GPT-OSS-20B-medium	1.56	1.57	1.43	1.21	1.36

Prompts

ProofBench (Strict) | ProofBench | GIMO | Simple | OPC

```
You are an **expert math proof grader**. You are judging the correctness of an
LLM-generated proof for a math problem.

### Input
Your input will consist of:
* **Problem Statement**: A mathematical problem that the proof is attempting to
solve.
* **Marking Scheme**: A problem-specific grading rubric (0-7 scale) with
checkpoints, zero-credit items, and deductions. You must follow this scheme when
assigning points.
* **Proof Solution**: The proof that you need to evaluate. This proof may contain
errors, omissions, or unclear steps. The proof was generated by another language
model.

### Task
Analyze the proof carefully.
**Core principles (in order of precedence):**
1) **Mathematical validity** of the proof's reasoning and conclusion.
2) **Problem constraints** (e.g., unique required final value; forbidden tools if
```

Figure 5. Prompt traces used for the proof-generation and evaluation pipeline.

We then compare the choice of grader models and evaluate whether including a reference proof alongside the marking scheme improves performance (Table 3). We conduct this experiment on the in-distribution grading benchmark as it is more representative of scenarios that the grader will encounter during training. We observe that the performance differences between models are minimal. GPT-OSS-20B with medium reasoning performs on par with the alternatives while being significantly cheaper and faster, so we adopt it as our grader for training. Including a reference solution slightly degrades performance, so we exclude it from the final grader configuration.

Table 3. Results on our in-distribution grading benchmark. Lower is better.

Model	ProofBench Strict	ProofBench Strict (with ref)
GPT-OSS-20B-medium	1.19	1.26
GPT-OSS-20B-high	1.17	1.19
GPT-OSS-120B-medium	1.16	1.24

Outcome-Reward RL with Long Response Lengths

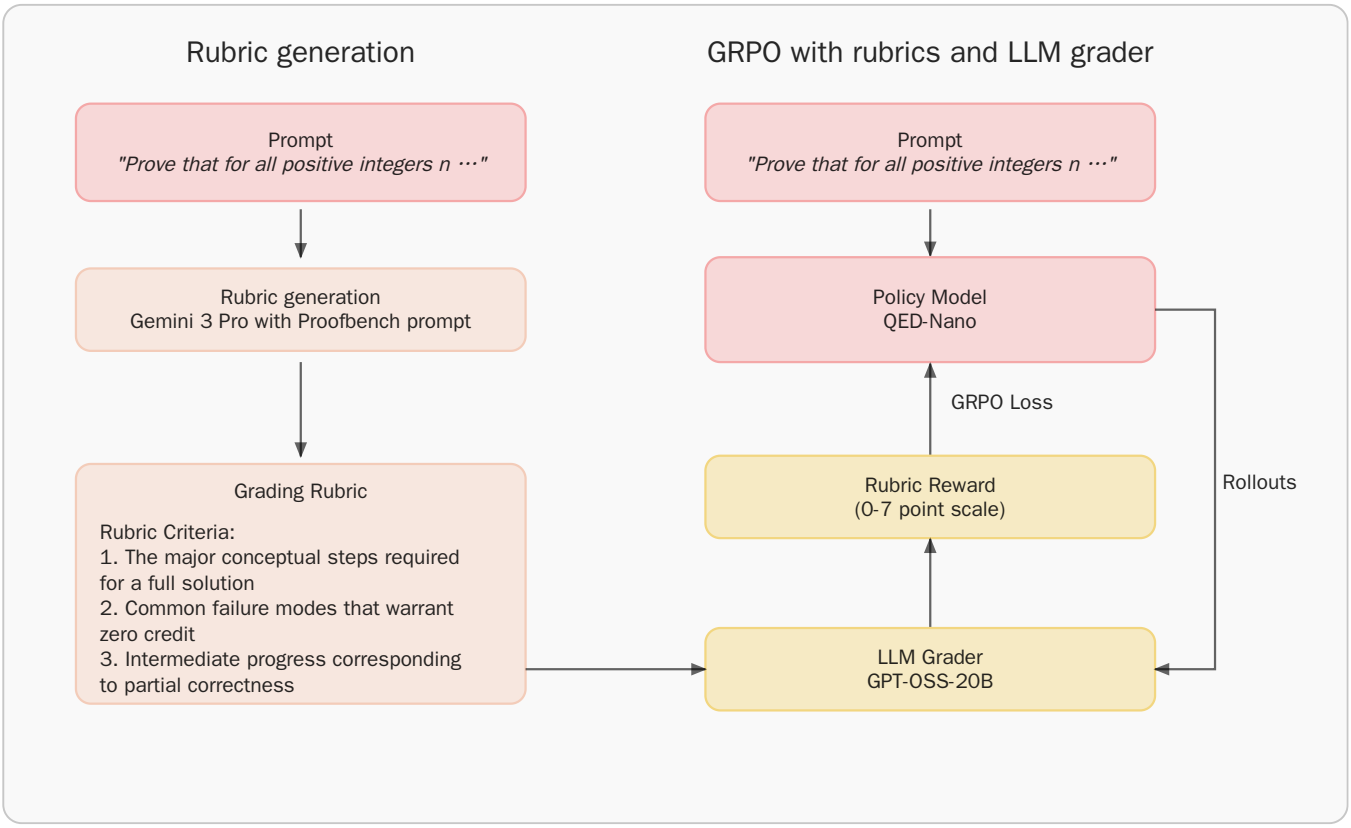


Figure 6. A schematic illustration of our pipeline for outcome-reward RL training of QED-Nano. We train with rubric-based rewards derived from a grading scheme as discussed in the Setup section.

Equipped with this grading scheme, we run RL to optimize the resulting outcome rewards. Two design choices remain when instantiating an RL run: the prompt set and the RL hyperparameters, in particular, the number of parallel rollouts per problem and the maximum response length. As discussed in the previous section, we construct a prompt set such that the base model's pass@1 scores follow a unimodal, heavy-tailed distribution (by modifying the distribution shown in Figure 7), with a peak near difficult problems and a decreasing probability of sampling substantially easier ones.

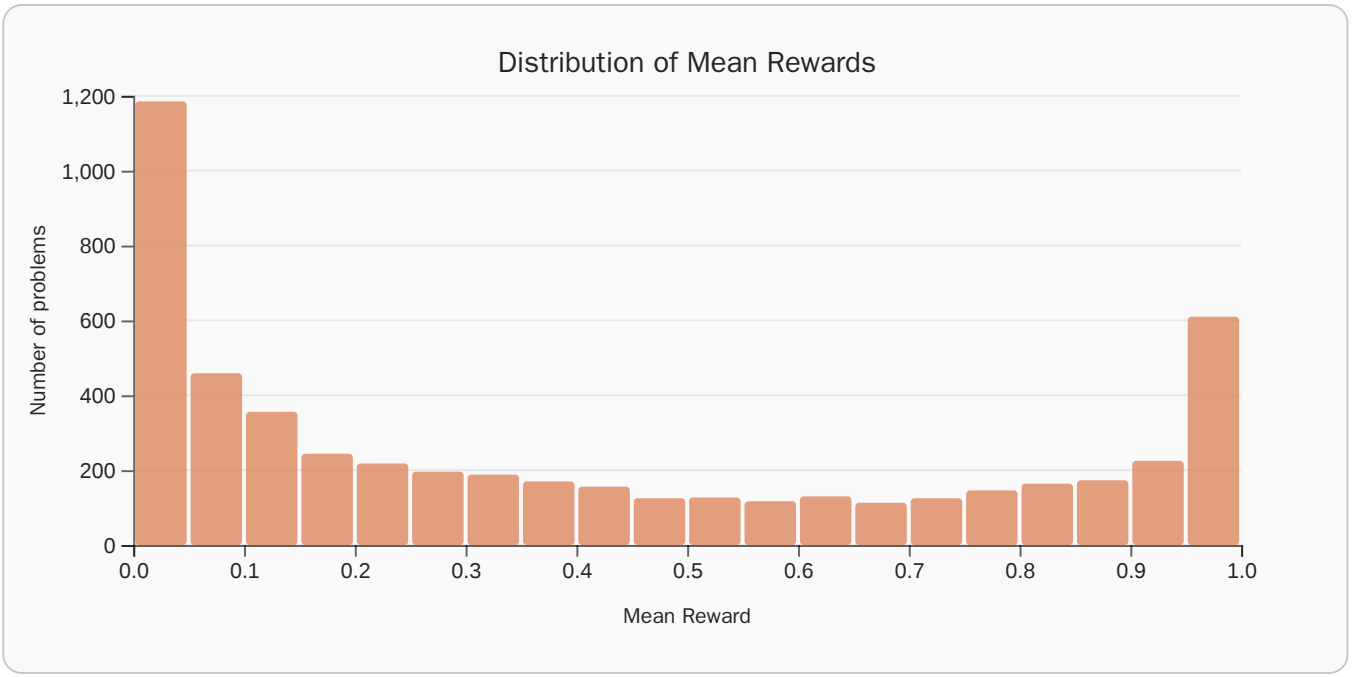


Figure 7. A schematic showing the distribution of the average reward per problem in our unfiltered prompt set. We remove all problems that attain a pass@1 score > 0.7 and use the remainder as our prompt set for training. We also remove problems where pass@1 score = 0.0.

We completely remove all very easy problems on which the base model can attain a pass@1 score higher than 0.7 and also remove the extremely hard problems. With this prompt set, we now describe our workflow for setting the various hyperparameters of the RL algorithm.

Base RL algorithm. We use GRPO as our base RL algorithm and build on PipelineRL to implement an asynchronous, streaming variant of this algorithm (Figure 8).

Figure 8. A schematic illustration of an asynchronous, streaming variant of GRPO that we also employ in our PipelineRL implementation. Image from the Magistral tech report

This implementation performs off-policy updates, with a maximum lag of 5 gradient steps between the current policy and the reference policy. We ablate several hyperparameters, including the number of parallel rollouts per problem, the entropy coefficient, and the KL divergence loss. We utilize an entropy coefficient of 1e-4 through training and no KL regularization. Consistent with prior work, we find that a larger number of rollouts n per problem improves performance when sufficient training epochs are run. Based on initial experiments with $n = 4, 8, 16$, we selected $n = 16$ because the fraction of problems on which no successful rollout is sampled is merely 2-3% at $n = 16$, which ensures a stable training signal (Figure 9). Running at this scale required 7 nodes to generate rollouts at a batch size of 64 problems (i.e., a total batch size of 1024 samples per step) and 4 nodes for the trainer.

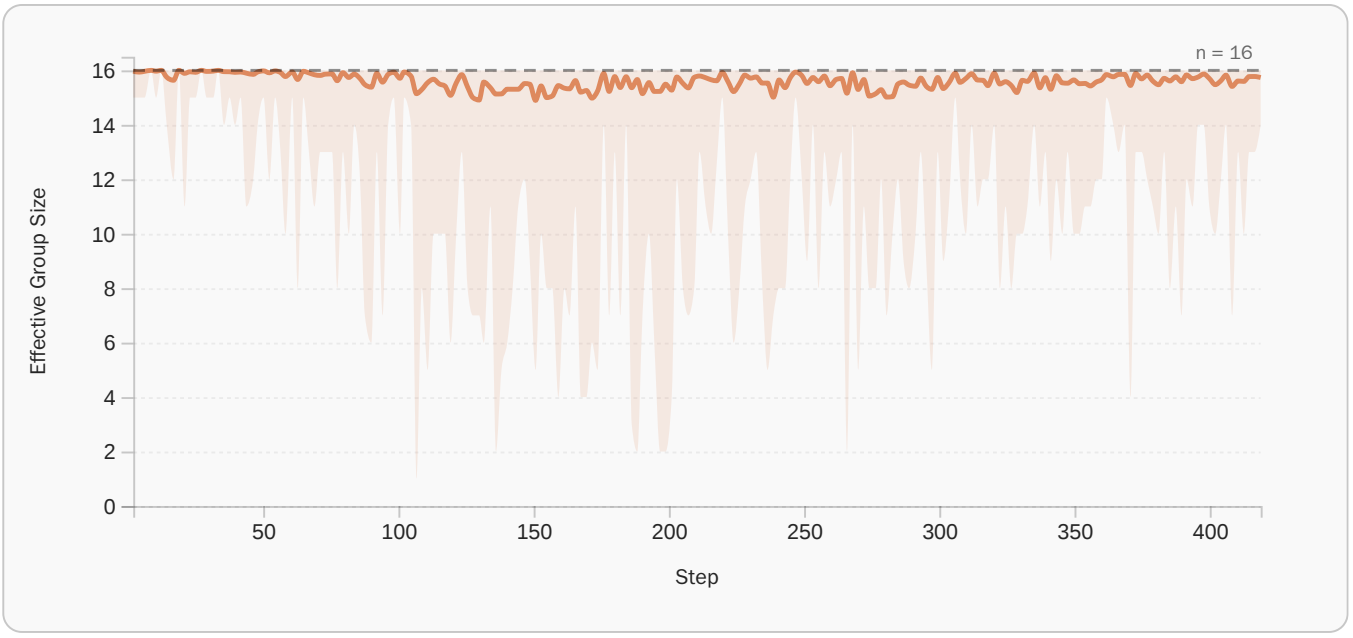
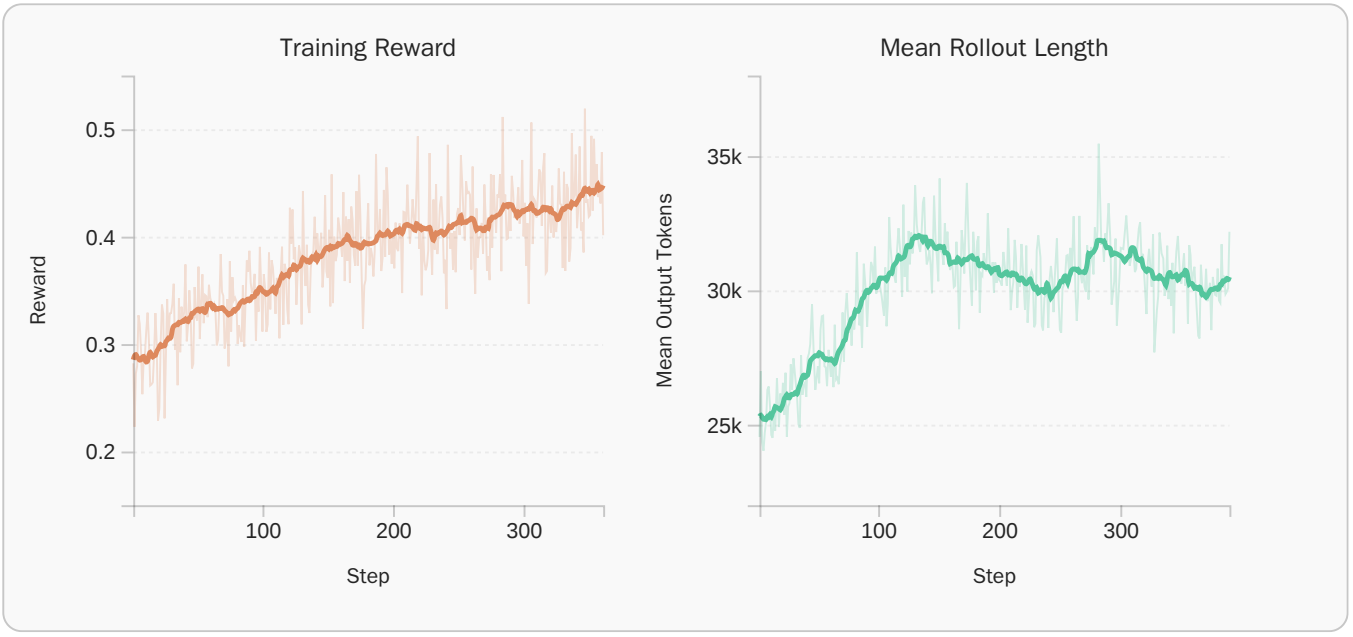


Figure 9. Effective group size throughout RL training. With n = 16 parallel rollouts per problem, the effective group size remains close to the maximum for most of training, indicating that nearly all problems receive both successful and unsuccessful rollouts — ensuring a stable training signal.

We set the maximum response length to 50,000 tokens for RL training, since 95% of responses from the base model terminate within this limit. As training progresses, however, we observe a noticeable increase in output length, consistent with observations from DeepSeek-R1 and others. A representative learning curve and corresponding evaluation scores are shown in Figure 10. We observe a noticeable increase in both the training and evaluation scores (on both IMO-ProofBench and ProofBench).



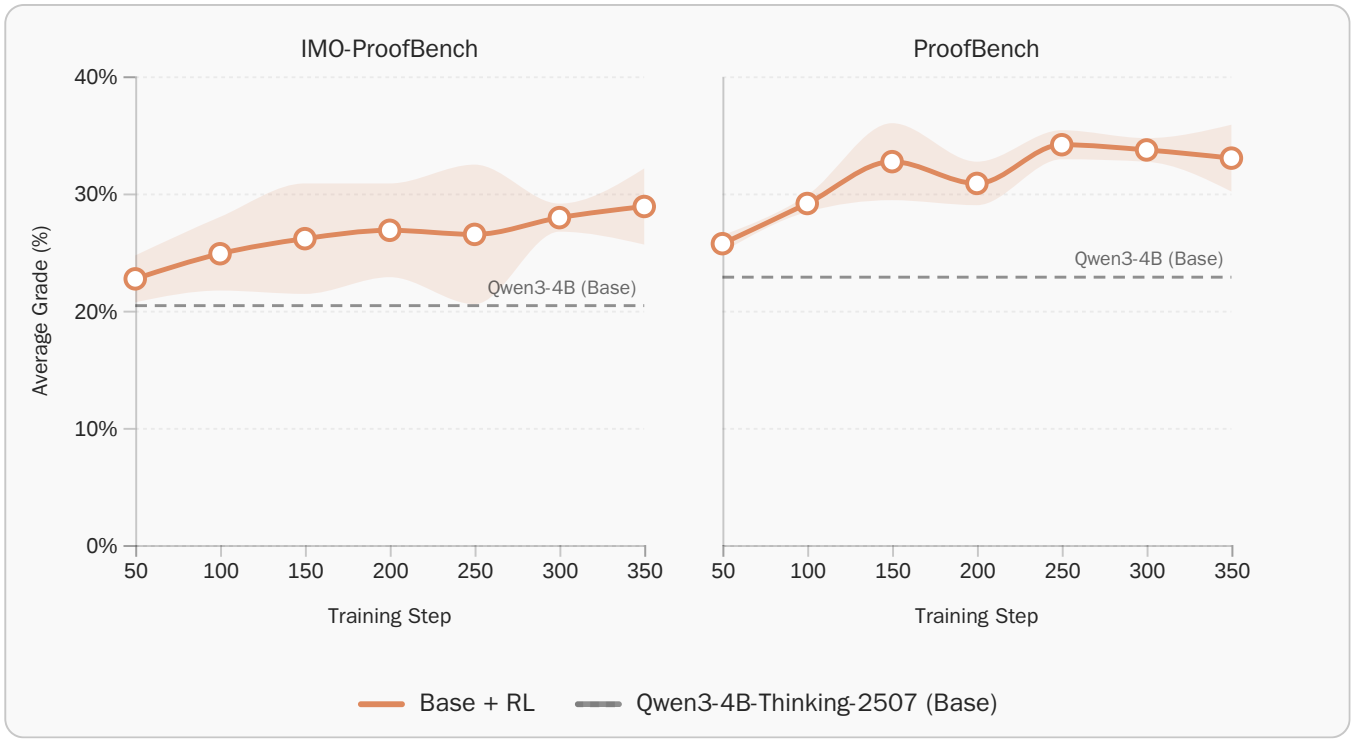


Figure 10. RL training curves with rubric-based rewards and corresponding evaluation metrics on IMO-ProofBench and ProofBench. Observe that as training proceeds, training rewards rise steadily and mean output length increases. Note that this is the mean output length on the training prompt set, which also includes some simpler problems on which the model is not able to exhaust the full token budget.

RL for Continual Improvement at Test Time via Reasoning Cache

Having established that RL improves both training reward and test-time performance under the grader, the natural next step is to scale these gains further. For a small 4B model, increasing test-time computation provides a direct mechanism for extracting additional performance. A naive approach would increase the maximum response length during RL training, but this introduces substantial infrastructure costs and exacerbates variance in long-horizon optimization.

Instead of training on extremely long monolithic responses, we introduce additional structure into the generation process. In particular, we adopt an iterative decoding procedure during training in which the model produces short reasoning segments that can be optimized with standard RL, while still encouraging improvements in long-horizon performance. We implement this idea using the Reasoning Cache (RC) framework. RC decomposes reasoning into multi-step refinement cycles. At each iteration, the model generates a partial reasoning trace, summarizes its progress into a compact short textual “state representation”, and conditions the next rollout on both the original problem and this summary (Figure 11). Each subsequent summarization step updates the previous summary with any information added in the current reasoning step. Then, we train the model with RL to improve its summary-conditioned generation capabilities. This structure allows

the model to effectively explore reasoning horizons equivalent to hundreds of thousands of tokens while maintaining smaller training rollout lengths.

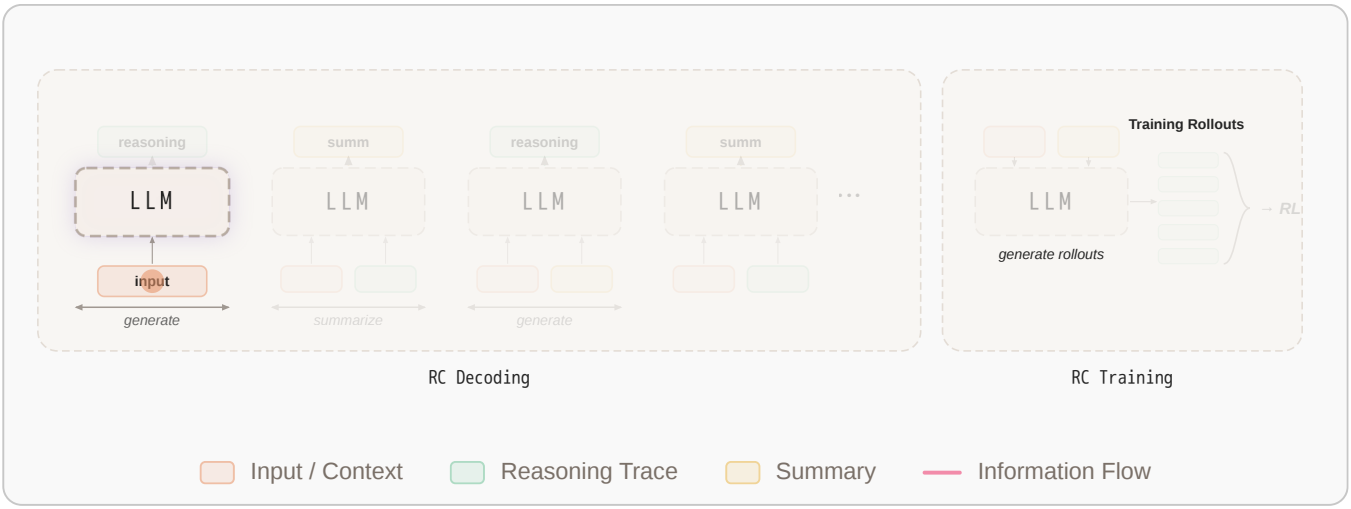


Figure 11. Illustration of the RC algorithm. RC decoding replaces standard autoregressive decoding at both train and test time. During RC decoding, the LLM generates a reasoning trace, summarizes it, discards the original trace, and conditions subsequent reasoning on this summary. This design decouples the effective reasoning horizon from the length of any single reasoning trace, thus maintaining tractable rollout lengths for outcome-reward RL while also enabling continual improvement at test time.

We apply RL updates across these RC states, training the model to improve conditioned on the summary. Empirically, RC improves training stability, convergence speed, and performance compared to standard RL (Figure 12).

Figure 12. Training curves comparing RL and RL with RC. Both runs use rubric-based rewards. RC achieves faster convergence and higher final reward, while rollout lengths grow more moderately under RC due to the iterative summarize-and-refine structure.

It also reduces the per decoding-turn response length, although this can easily be compensated for by running for more turns. Each subsequent turn improves over the average reward attained by

the previous turn (Figure 13).

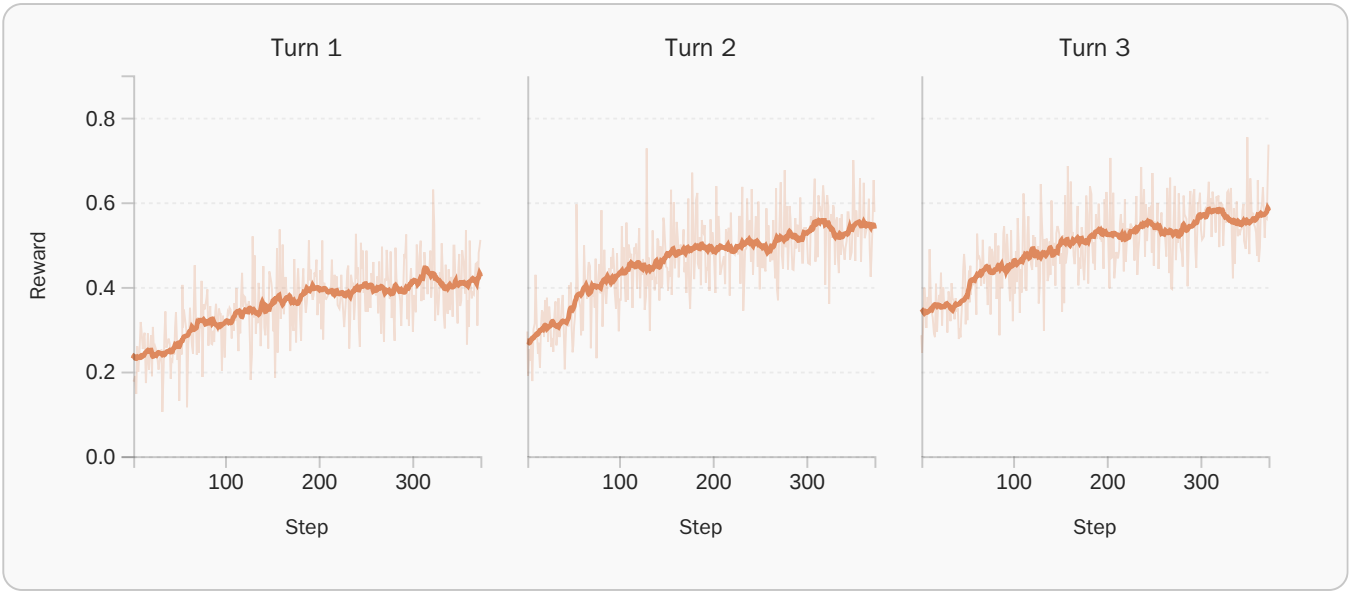


Figure 13. Per-turn mean reward during RC training. Each panel shows the reward for a successive reasoning-cache turn. The model improves with each additional turn, confirming that RC training teaches the model to refine its reasoning conditioned on prior summaries.

While we use the same model for both reasoning and summarization at test time, during training, we choose to avoid using a thinking model for summarization to speed up the training process. Instead, we use a frozen snapshot of the Qwen3-4B-Instruct-2507 model for summarization. That said, we observe that these sorts of gains with RC persist even when the same model performs both reasoning and summarization, suggesting that the primary benefit arises from extending the effective reasoning horizon rather than from any new information or prompt tuning.

Upon evaluation, we find that both the RL-trained and RC-trained models achieve similar performance within a single decoding turn. However, the RC-trained checkpoint improves substantially more when run with the RC scaffold (see Figure 13 below). In particular, the RC-trained model outperforms the RL-trained model at every turn, with the largest gap appearing within the first three turns, which matches the number of turns used during training. We also evaluate both RL- and RC-trained models using a different agentic scaffold, namely the DeepSeek-Math-V2 scaffold discussed later, and again observe larger gains for the RC-trained model. These results suggest that RC-style training better prepares the model to benefit from test-time scaffolds, and therefore we adopt RC training in our final recipe. An example of the scaffold is shown in Figure 14.

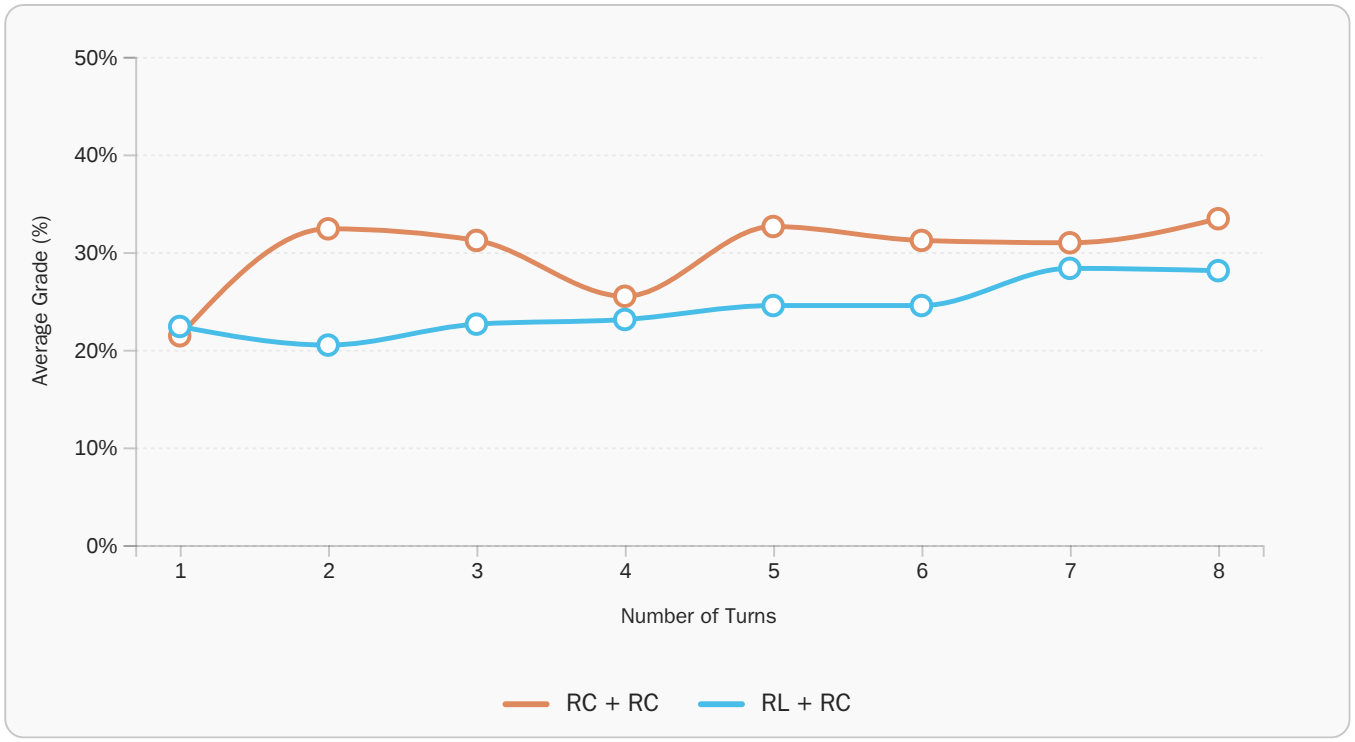


Figure 13. Average grade (normalized to 0–100%) on IMO-ProofBench as a function of reasoning-cache turns. Observe that applying the RC scaffold at test time on top of the RC-trained model attains higher performance than applying the RC scaffold on top of the RL-trained model. The gains are largest at turn 3 of the RC decoding process, which represents the number of turns also used for RC training.

Reasoning Cache Example

Reasoning 0 (t=0) | Summarization 0 (t=0) | Reasoning 1 (t=1)

Summarization 1 (t=1) | Reasoning 2 (t=2)

User — 149 words | Assistant (CoT) — 4,854 words | Assistant — 949 words

User

You are given a maths problem. You may also be given a summary of a previous attempt to solve it. This previous attempt may or may not be correct.

PROBLEM
Generate a rigorous proof to the following question:

Determine all functions $f : \mathbb{Z} \to \mathbb{Z}$ such that, for all $x, y \in \mathbb{Z}$, we have

$$f(2x) + 2f(y) = f(f(x + y)).$$

SUMMARY OF PREVIOUS ATTEMPT
[No previous summary]

INSTRUCTIONS
If no summary of a previous attempt is provided, solve the problem from scratch.

Figure 14. A multi-step reasoning dialogue showing chain-of-thought and summarization.

Initialization via Supervised Fine-Tuning

Despite the promising results from RL on top of the 4B base model, we found that building coverage over certain proof-writing strategies with an initial supervised fine-tuning stage provides

a better initialization for the RL run. Therefore, in parallel, we iterated on SFT for the base model. Our SFT recipe fine-tunes the base model on problems paired with proof solutions generated by deepseek-ai/DeepSeek-Math-V2, a 685B model fine-tuned specifically for Olympiad math (with a complex training procedure that involves meta-verifiers). We distill this teacher's reasoning traces into a compact dataset of $\approx$7.5k sampled responses suitable for fine-tuning our 4B base model. We describe this in detail below.

SFT dataset generation using DeepSeek-Math-V2. We generate solutions for problems in our curated dataset using a 128k-token context limit. Given the large size of the teacher (685B parameters), simply running inference on the teacher was challenging, and we had to orchestrate inference across 8 parallel instances (with SGLang), each distributed over two 8xH100 nodes (with TP=8, EP=8, PP=2). A central router load-balanced all inference requests, achieving a throughput of $\approx$3000 tokens/s. We first filter raw generations to retain only structurally valid completions containing closed reasoning blocks and explicit proof sections. We grade the solutions with Gemini 3 Pro and intentionally avoid discarding low-scoring samples simply because they might still provide useful information about proof-writing. This process yields a dataset of 7.5k proof-style responses across 4,300 distinct problems spanning Algebra, Calculus, Combinatorics, Geometry, Inequalities, Logic and Puzzles, and Number Theory.

We fine-tuned our base 4B model on this dataset using a global batch size of 32 for five epochs. We applied a cosine learning rate schedule with a 10% warmup and a peak value of 3×10^{-5}, which provided stable convergence while reducing validation SFT-loss on a hold-out set.

Data ablation: quantity vs. uniqueness. We performed several ablations with different data mixtures; we highlight the comparison between training on the full corpus of 7,500 prompt-completion pairs versus a strictly filtered set of 4,300 correct solutions, where one solution is associated with a unique problem. We find that training on only the unique problems achieves a higher final performance on IMO-ProofBench. The checkpoint at step 372 of this run was therefore used as an initialization for RL training (Figure 15).

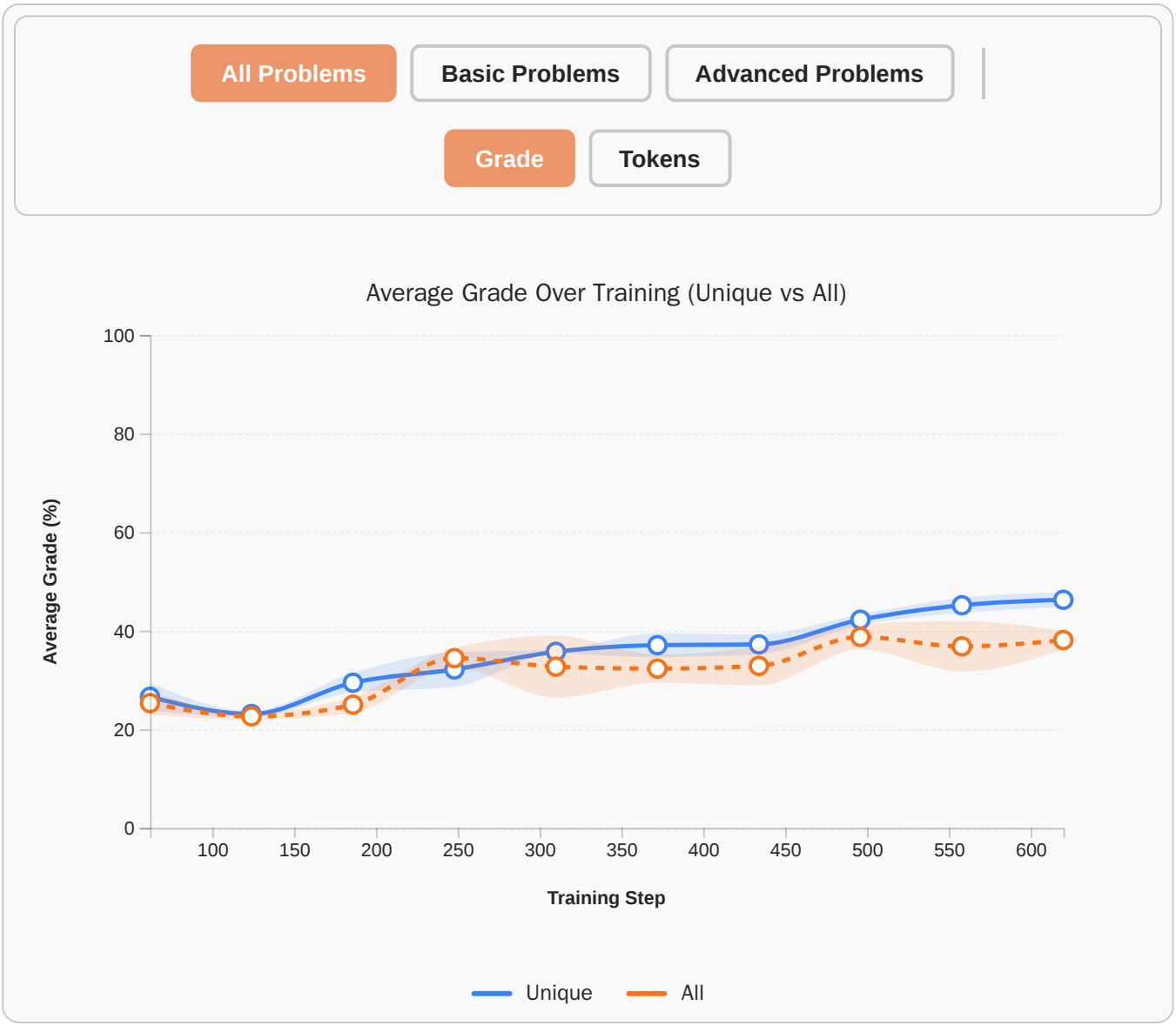


Figure 15. As illustrated in the figure above, using the unique dataset achieved higher performance on IMO-ProofBench. We use this initialization for our RL runs.

Challenges and limitations of SFT. SFT serves as a strong bootstrap and produces a clear improvement over the base 4B model. However, the process also introduced significant drawbacks, most notably length explosion. Although the training data caps sequences at 45k tokens, the fine-tuned model frequently generates outputs that grow to hundreds of thousands of tokens, and are typically much longer for incorrect proof attempts. Rather than producing structured long-form reasoning, the model often imitates the surface appearance of extended proofs, repeating or meandering until the context window is exhausted. This behavior is a natural consequence of offline training on data that comes from a bigger model (or data that is generally "hard to fit") and indicates the need for a more "experiential" learning paradigm instead.

RL provides a natural mechanism for experiential learning, and in practice, we observe that response lengths decrease following RL training. This trend also holds when training with RC. However, the early phase of RL is heavily confounded by rollout truncation and a high overflow rate (often around 60% on average), which impairs credit assignment and reduces the effectiveness of

RL and RC when initialized from SFT. An immediate direction for future work is to address this length overflow issue more directly. One possibility is to replace SFT with on-policy distillation, though this is computationally expensive due to the inference costs of the 685B-parameter DeepSeek-Math-V2 model. A more practical alternative is to approximate it by blending in on-policy traces during SFT. A complementary approach is to introduce a curriculum during RL: first training on problems that do not suffer from severe overflows, thereby enabling the model to realize the benefits of RL before scaling to longer-horizon settings.

> 👨‍🍳 Final Recipe
>
> Our final recipe consists of an initial SFT step to imbue the model with the ability to write high-quality proofs. Then, we perform rubric-based RL training with the reasoning cache approach to make the model capable of effectively thinking longer when used with test-time scaffolds. Finally, we deploy the trained model with a test-time scaffold.

Further Scaling Test-Time Compute with QED-Nano

We next explore whether scaffolds that combine parallel and sequential generation can further scale token usage and thereby performance. Since our RL training procedure optimizes summary-conditioned generation without enforcing a specific summary format, we expect that the trained model would benefit from a broad class of scaffolds, including those that condition subsequent generations on detailed verification logs of the previous generations. Indeed, the trained model does choose to verify information present in the summary. We therefore evaluate several alternative test-time scaffolds to further scale the performance of QED-Nano.

In particular, we explored the following scaffolds:

- Reasoning-Cache (RC): The decoding algorithm that we used for RL post-training, which iteratively summarizes the current attempt and conditions subsequent response generation on it. This approach does not utilize parallel sampling directly and we only run it for 3 turns in this result.
- Self-Check: A simple generate-verify-improve loop that ends when the verifier cannot find any flaws in the solution anymore.
- Nomos: This scaffold first generates n solutions and verifies each of them once. It then filters the solutions to only keep the k best ones, after which it is run through a single "consolidation" stage, where the model is presented with all k solutions and asked which

group of solutions is most likely to be correct. Among that group, the agent runs a simple knockout tournament with an LLM-judge to select the best one.

- RSA: This scaffold first generates n solutions. In each subsequent stage, it then generates n new solutions by randomly conditioning each new solution on k existing ones. After several iterations, a random proof is selected. We make one minor improvement over the original design: instead of selecting an arbitrary proof, we run a knockout tournament with an LLM-judge on the solutions from the last stage.
- DeepSeek Math (DSM): This scaffold first generates n solutions, each of which is self-evaluated n times. Solutions are sorted by their average self-evaluated score, and the top n solutions are improved by presenting the model with the solution and some of the feedback generated by the self-evaluation stage. These new solutions are added to the solution pool. This process is iterated several times before the solution with the highest overall score across all iterations is returned.

In preliminary experiments on RL-trained checkpoints initialized from the base model, we evaluated several test-time scaffolds. DSM, RSA, and RC consistently yielded the strongest improvements, with DSM outperforming RSA in several settings, while other scaffolds provided more modest gains. Among these, RC does not run parallel sampling at each turn, making it convenient for training but suboptimal for maximizing test-time performance, whereas DSM can effectively scale parallel compute as well. We therefore adopted DSM for our main experiments. This choice was further motivated by the fact that our SFT initialization leveraged traces from DeepSeek-Math-V2, which is explicitly trained for self-verification. Consequently, we hypothesized that a scaffold that explicitly incorporates self-verification would be particularly effective for our final trained QED-Nano model. Our main results confirm this hypothesis.

Quantitative evaluation. Figure 16 summarizes the performance of different scaffolds applied to our final QED-Nano model (note that the preliminary experiments above were done on the RL-trained checkpoint from the base model). RC yields the smallest performance gain within 3 turns (which is perhaps expected), but it also requires only about twice the number of tokens. In contrast, RSA and DSM provide substantial improvements of 17% and 14%, respectively. However, they are substantially more expensive: RSA costs about 20 times as much as the base model, and DSM about 16 times as much. Overall, DSM provides a reasonable tradeoff for performance within a given token budget and we therefore utilize it. However, one could also choose to use RSA given these results.

Agent Scaffold Comparison: Average Grade vs Average Tokens

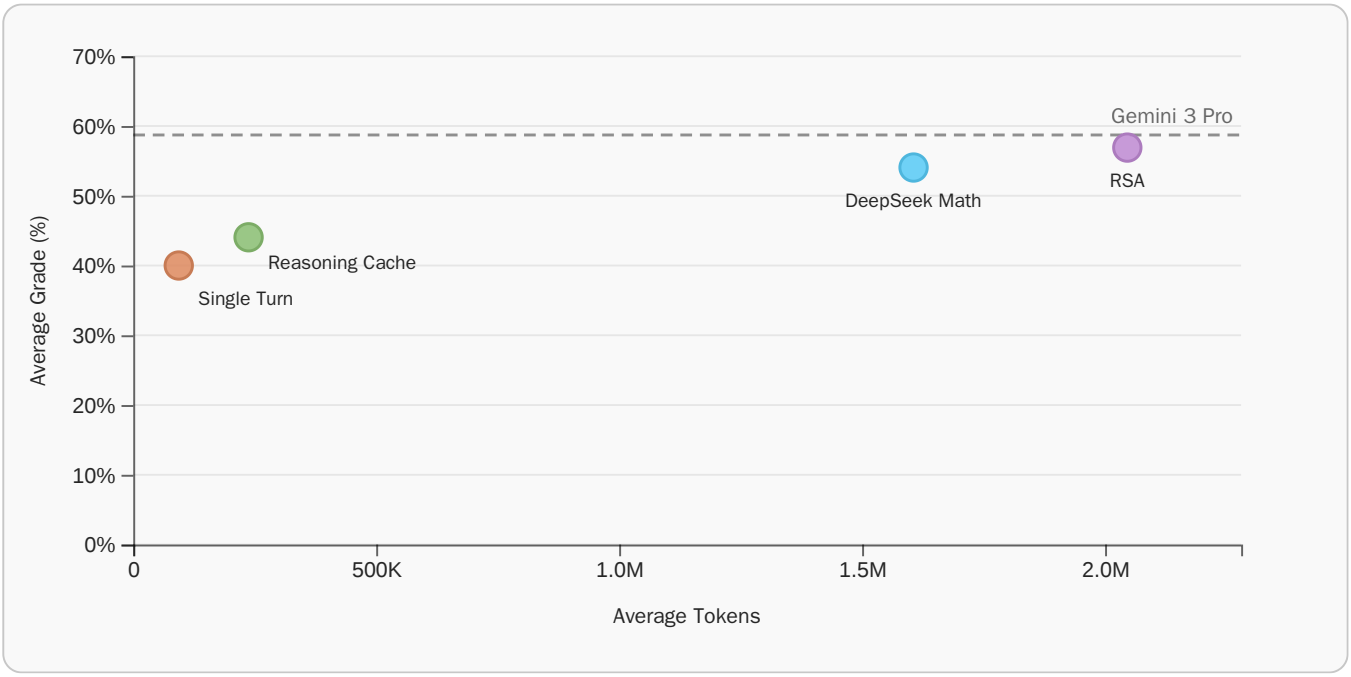


Figure 16. Comparison of different agent scaffolds applied on the QED-Nano model and evaluated on IMO problems, showing the trade-off between token usage and performance.

Qualitative Analysis of Solutions from QED-Nano

We manually examined a subset of QED-Nano's generated proofs to assess two additional dimensions beyond benchmark scores: (1) whether the model attempts to reward-hack the LLM-based grader, and (2) the intrinsic quality of the proofs themselves. An experienced human evaluator from our team, with a substantial background in evaluating LLM-generated mathematical proofs, reviewed a sample of proofs and compared their judgments with those of the automated grader. Detailed annotations are provided in the accompanying figure; below, we summarize the main observations.

Agreement between LLM judge and human judgment. We found no clear evidence of reward hacking on our Gemini 3 Pro grader. The human evaluator agreed with the automated grader on most problems, though the LLM grader was occasionally a bit more generous. Only one problem showed a significant change in score: the QED-Nano agent's solution to IMO 2025 Q2. Our human grader judged the heavy computational approach incorrect, noting an algebraic error and too many gaps for a fully rigorous proof. We do not attribute this discrepancy to deliberate reward hacking, as other models generally attempted similar approaches to Q2.

Proof quality. The generated proofs are generally well structured and logically organized, making them easy to read and follow. The most prominent weakness is a consistent preference for computation-heavy approaches. In geometry problems in particular, the solutions always rely on

coordinate or algebraic arguments (“bashing”) rather than synthetic arguments. In other domains, the model also frequently provides proofs that are more computational than the human-written ground-truth solution. However, this tendency is not unique to QED-Nano as it reflects a broader pattern across LLMs (based on prior experience from evaluating various frontier models on MathArena evaluations).

Agentic vs. base model. We compared proofs generated by the base model with those produced with the DSM agent. In most cases, the outputs are stylistically and structurally similar. The primary benefit of the agentic setup appears in cases where the base model’s solution contains a clear but nontrivial mistake. There, the agent often produces a correct proof that addresses that particular mistake.

IMO 2025. We explicitly evaluated our model on the 2025 IMO problems. According to the Gemini 3 Pro judge, QED-Nano achieves a score of 22/42 when used with the test-time scaffold. The standalone QED-Nano model achieves 12/42 under the same judge. We observed, however, that the trained model does make fairly simple algebraic or transcription errors, such as mismanipulating terms or incorrectly copying previously derived expressions. The Gemini 3 Pro judge is relatively lenient toward these mistakes, and is also too likely to give intermediate points for trivial steps. According to our human grader, QED-Nano attains 14/42 with the scaffold and 7/42 without it.

Although some of these errors would very likely be mitigated with additional training or larger models that tend to follow the semantics of language or by employing a stricter grading protocol during training, the results also make clear that the model struggles substantially on some of the most challenging problems, where it is unable to make meaningful progress.

Comparison with other models. We also compared QED-Nano’s proofs with those generated by the Qwen-3-4B-Thinking-2507 base model and Nomos-1. The base model’s outputs are of very poor proof quality. It is clearly trained to obtain the correct final answer while disregarding mathematical rigor. While it occasionally finds the correct answer to a problem, its proofs often contain obvious mistakes, such as circular arguments and simplifying assumptions. Nevertheless, our recipe is able to improve this base model to produce proofs of much higher quality.

Nomos-1 has a significantly different style compared to QED-Nano. Its proofs are more concise and direct, more similar to expert human-written solutions. For instance, it does not repeat the problem statement and instead immediately provides the answer, often proves lemmas as separate statements, frequently skips computational steps, and writes in a very brief, matter-of-fact statements. While expert-written solutions can generally be trusted to be this compressed, the general sycophancy found in LLMs and their tendency to hide mistakes in proofs, makes this behavior more problematic. Determining which writing style is better is quite subjective, but we encourage you to read some of the proofs below to decide for yourself!

Vibe checks. We evaluated the model with a small set of non-mathematical prompts to confirm it retained broader capabilities. Remarkably, it still follows instructions effectively, with no noticeable regressions in non-mathematical reasoning tasks.

PB-Basic-001 | PB-Basic-004 | PB-Basic-005 | PB-Basic-024 | PB-Basic-028

PB-Advanced-013 | PB-Advanced-025 | PB-Advanced-030 | IMO-2025-Q1

IMO-2025-Q2 | IMO-2025-Q3 | IMO-2025-Q4 | IMO-2025-Q5 | IMO-2025-Q6

Problem

Determine all functions $f : \mathbb{Z} \to \mathbb{Z}$ such that, for all $x, y \in \mathbb{Z}$, we have

$$f(2x) + 2f(y) = f(f(x + y)).$$

QED-Nano (7/7) | QED-Nano (Agent) (7/7) | Qwen3-4B-Think (1/7)

Nomos-1 (7/7)

Model Summary

Model: QED-Nano

Grade: 7/7

Dataset: lm-provers/imoproofbench-outputs

Config: hf-imo-colab-qwen3-4b-thinking-2507-proof-rc_v0900-step-000150-google

Split: 20260203_133642

Comment

The proof is correct and very readable. It explicitly derives the key identities early (e.g., substituting special values to obtain $f(2x) = 2f(x) - c$).
It fully expands its algebraic manipulations, maybe a bit too extensively, but this makes verification easy despite the extra length.

Proof

Solution

Let $f : \mathbb{Z} \to \mathbb{Z}$ satisfy

$$f(2x) + 2f(y) = f\big(f(x+y)\big) \qquad (\forall x, y \in \mathbb{Z}). \tag{1}$$

---

1. First consequences

Put $x = 0$ in (1):

$$f(0) + 2f(y) = f\big(f(y)\big) \implies f(f(y)) = f(0) + 2f(y). \tag{2}$$

Figure 17. Per-problem model proofs with reviewer comments and grades.

Discussion and Conclusion

In this blog, we introduce QED-Nano to demonstrate that Olympiad-level problem-solving is not reserved for frontier-scale models with 100B+ parameters. With a post-training recipe that first instills high-quality proof-writing strategies via SFT and then applies RL to explicitly optimize long-horizon improvement through test-time scaling, a 4B model can produce substantially stronger proofs than its base initialization and compete with much larger open models when paired with additional test-time compute. On IMO-ProofBench, our open-source QED-Nano (Agent) closes much of the gap to Gemini 3 Pro while being at least 3x cheaper to run and requiring significantly lower training costs. This highlights a practical path toward strong reasoning through specialization and test-time adaptation rather than scaling only parameter count. Averaged across benchmarks, QED-Nano (Agent) also significantly outperforms larger open models such as Nomos-1 and GPT-OSS-120B.

We also outline our end-to-end workflow to help others train small models for effective long-form reasoning. More broadly, our recipe is simple and applicable to other domains where outcomes are difficult to verify directly but structured rubrics can be constructed. We did not observe evidence of reward hacking under rubric-based rewards, further supporting the robustness of this

approach. To encourage follow-up work, we release our models, datasets, and code implementations.

Going forward, several avenues could further improve QED-Nano. The most immediate action items include improving the synergy between SFT and RL. In particular, mitigating the length explosion introduced by SFT early on would likely amplify the gains from subsequent RL by speeding up the process of credit assignment. A second short-term direction is to refine the grader design, for example, by gradually tightening rubric penalties as training progresses or by using strict reward designs, thereby incentivizing increasingly rigorous and polished proofs as the model learns to make progress. Finally, incorporating hints or guidance during training, such as conditioning on plans from oracle solutions or the grading scheme, will help the model tackle harder problems during training and enable further scaling of RL to achieve stronger results.

Beyond these goals, more fundamental questions remain. One direction is developing approaches that imbue the LLM with the ability to synthesize genuinely novel ideas or “aha” insights when solving the hardest problems. Like most LLMs, QED-Nano tends to rely on computation-heavy approaches rather than identifying elegant structural insights early on. This reflects the style of reasoning RL optimizes models for. Designing scalable training paradigms that encourage broader exploration of reasoning strategies, rather than refinement of a single computational path, is therefore an important challenge. From a workflow perspective, another key direction is to develop methods that more directly optimize for the specific test-time scaffolds used at deployment, tightening the alignment between train-time objectives and inference-time behavior. We encourage the community to study these aspects.

Author Contributions

This is a team effort with members from CMU, Hugging Face, ETH Zurich, and Numina.

Our team members (in alphabetical order) are as follows:

- CMU: Aviral Kumar, Yuxiao Qu, Amrith Setlur, Ian Wu
- Hugging Face: Edward Beeching, Lewis Tunstall
- ETH Zurich: Jasper Dekoninck
- Project Numina: Jia Li

All members contributed to the project substantially. Specifically:

- Yuxiao Qu developed the initial version of the grader and verifier-based RL approach, built the grading-scheme pipeline, curated and processed the proof datasets, and implemented the

initial reasoning-cache and multi-turn training loops that started us in this direction. With Amrith Setlur and Lewis Tunstall, he ran a number of ablations that informed the final RL runs.

- Amrith Setlur adapted the PipelineRL infrastructure for the verifier-based RL approach, optimized RL configurations for stability and scale, implemented the asynchronous and streaming reasoning-cache RL training infrastructure, and proposed several ablations and algorithmic strategies for the training runs. With Yuxiao Qu and Lewis Tunstall, he ran a number of ablations that informed the final RL runs.
- Ian Wu, as primary author of the Reasoning Cache method, provided core technical guidance on RC experimentation, evaluation, and training pipelines, which shaped how it was utilized throughout the project.
- Edward Beeching led several evaluations, developed the synthetic data generation pipeline for DeepSeek-Math-V2, and ran the SFT ablations to analyse model length-control behavior and training dynamics.
- Lewis Tunstall led the large-scale RL infrastructure efforts, benchmarking and stabilizing multiple RL frameworks at the start of the project, optimized inference throughput, and ran the largest training and evaluation experiments. With Amrith Setlur and Yuxiao Qu, he ran a number of ablations that informed the final RL runs. He advised several other aspects of the project.
- Jasper Dekoninck led the benchmark design and ensured rigorous evaluations, benchmarked several test-time agent scaffolds and developed our final scaffold, built the IMO-ProofBench and ProofBench splits, filtered training datasets and created the grading schemes for them, designed the benchmarks for the RL grader, and led extensive model-based and human evaluations to ensure robustness and correlation with human proof quality.
- Jia Li curated and expanded high-quality AoPS and Olympiad datasets, developed grading-scheme generation workflows, and explored scalable problem synthesis and verification strategies in the project initially.
- Aviral Kumar advised the overall project and contributed to the ideas behind long-horizon training, curriculum design, reward formulation and some ideas on data construction.

Acknowledgements

We thank Leandro von Werra, Andres Marafioti, Thibaud Frere, Graham Neubig, Sewon Min, Wenjie Ma, and Katerina Fragkiadaki for helpful discussions and feedback. AS, YQ, IW, and AK

thank the FLAME center at CMU, the DeltaAI cluster, and the NAIRR program for providing GPU resources that supported a part of the experimental iteration. We thank Google Cloud for Gemini 3 Pro API credits. AS and AK thank the Laude Institute Slingshots program for support and feedback, and Braden Hancock and Andy Konwinski at the Laude Institute for discussions and feedback. EB and LT thank Hugo Larcher and Mathieu Morlon for keeping the GPUs running hot on the Hugging Face cluster 🔥.

Citation

For attribution in academic contexts, please cite this work as

```
LM Provers Team (2026). "QED-Nano: Teaching a Tiny Model to Prove Hard Theorems".
```

BibTeX citation

```
@misc{qednano2026,
  title        = {QED-Nano: Teaching a Tiny Model to Prove Hard Theorems},
  author       = {LM-Provers and Yuxiao Qu and Amrith Setlur and Jasper Dekoninck and Edward
Beeching and Jia Li and Ian Wu and Lewis Tunstall and Aviral Kumar},
  year         = {2026},
  howpublished = {https://huggingface.co/spaces/lm-provers/qed-nano-blogpost},
  note         = {Blog post}
}
```

Reuse

Diagrams and text are licensed under CC-BY 4.0 with the source available on Hugging Face, unless noted otherwise.

Made with ❤️ with research article template