RELIABLE FINE-GRAINED EVALUATION OF NATURAL LANGUAGE MATH PROOFS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in large language models (LLMs) for mathematical reasoning have largely focused on tasks with easily verifiable final answers while generating and verifying natural language math proofs remains an open challenge. We identify the absence of a reliable, fine-grained evaluator for LLM-generated math proofs as a critical gap. To address this, we propose a systematic methodology for developing and validating evaluators that assign fine-grained scores on a 0–7 scale. Our approach first constructs a carefully designed, problem-specific marking scheme, and then uses it as a foundation to systematically study other key design choices, including the backbone model, additional context, instruction sets, and evaluation workflows. To enable this study, we introduce PROOFBENCH, the first expert-annotated dataset of fine-grained proof ratings, spanning 131 problems from major math competitions and 393 LLM-generated solutions (from o3, Gemini 2.5 Pro, and DeepSeek-R1) with expert gradings. Our evaluation shows that a strong reasoning backbone, a detailed marking scheme, and simple ensembling are crucial for high performance. This leads to our best evaluator, PROOFGRADER, which achieves an RMSE of 1.093 compared to expert grading, significantly outperforming simpler baselines. Furthermore, to demonstrate its practical utility, we test PROOFGRADER as a reward model in a best-of-n selection task. At $n = 8$, it achieves an average score of 4.05/7, bridging more than 90% of the performance gap between a naive binary evaluator (2.59) and the human oracle (4.21), underscoring its potential to improve downstream proof generation.

1 INTRODUCTION

Large language models (LLMs) have recently achieved remarkable progress in mathematical reasoning, attaining strong performance on a variety of benchmarks. Such models are especially strong at solving final-answer problems because they can be trained using reinforcement learning against simple answer verifiable rewards (Shao et al., 2024; DeepSeek-AI et al., 2025; Yang et al., 2025; Yu et al., 2025; Wang et al., 2025). However, these methods do not transfer to proof generation for two reasons: (i) many proof problems do not admit a single, easily checkable final answer; and (ii) even when a final answer exists, verifying it is insufficient to assess proof validity, as the reasoning may contain substantial intermediate errors (Petrov et al., 2025; Dekoninck et al., 2025). Because proof-generation tasks constitute a large share of mathematical problem solving in research and education, this necessitates reliable proof evaluation methods.

We identify reliable proof evaluation as a key bottleneck for improving proof generation because it is essential for providing faithful assessments of model capabilities and accurate reward signals for training models. Existing methods are clearly insufficient: human grading, while accurate, is slow and costly, especially for training; automatically translating natural-language proofs into formal languages (e.g., Lean) for machine checking is brittle and remains an unsolved, extremely challenging research frontier (Gao et al., 2025; Liu et al., 2025). While the "LLM-as-a-judge" (Zheng et al., 2023a; Sheng et al., 2025; Dekoninck et al., 2025; Huang & Yang, 2025) paradigm is promising, its application to mathematics is unsettled, and outcomes are sensitive to evaluator design—model choice, available context, rubric construction, and prompting—without clear guidance on which choices matter.

To address this, we conduct a comprehensive study of proof-evaluator design. To support this analysis, we establish **PROOFBENCH**, the first expert-annotated dataset for fine-grained proof evaluation that spans problems from multiple contests and years. It contains 393 LLM-generated solutions to 131 problems from major math competitions (EGMO, USAMO, IMO, USA TST, APMO and PUTNAM), produced via a two-stage process: first, problem-specific marking schemes are generated to ensure consistency; then, human experts use these schemes as a guide to score proofs, while allowing for valid alternative solutions. Following a principled evaluation protocol, we explore and quantify the impact of different backbone models, context components (such as reference solutions and problem-specific marking schemes), and instruction sets. We also investigate more advanced techniques, including ensembling multiple evaluation runs to improve robustness and staged workflows that decompose the complex evaluation task into simpler, sequential steps.

Our analysis yields **PROOFGRADER**: an LLM-based evaluator integrating a strong backbone with informative context (both reference solutions and a marking scheme) and simple ensembling. This design achieves a low Root Mean Square Error (RMSE) of 1.093 against expert scores, significantly outperforming simpler baselines. We further demonstrate its practical utility by testing it as a reward model in a best-of-n selection task. For $n = 8$, PROOFGRADER selects proofs with an average expert score of 4.05/7, closing over 90% of the performance gap between a naive binary evaluator (2.59) and the human oracle (4.21).

Key Contributions. To summarize, our work makes the following contributions:

- A systematic comparison of evaluator design factors with quantitative guidance on what drives better alignment with expert judgments.
- We will open source PROOFBENCH, an expert-graded benchmark dataset with problem-specific marking schemes for reproducible, fine-grained proof evaluation
- We introduce PROOFGRADER, the high-performance evaluator emerging from our study, which achieves strong alignment with experts and demonstrates its value as a reward signal for downstream tasks.

2 RELATED WORK

Benchmarks for Mathematical Reasoning. Datasets such as GSM8K (Cobbe et al., 2021), which targets grade-school word problems, and MATH (Hendrycks et al., 2021) and AIME, which extend coverage to algebra, calculus, and contest problems, have served as important benchmarks in math reasoning, but primarily focus on short, closed-form answers. More recent competition based benchmarks, including Omni-MATH (Gao et al., 2024), OlympiadBench (He et al., 2024), HARP (Yue et al., 2024), and MathArena (Balunović et al., 2025), are substantially more challenging; however, they still largely emphasize answer matching or other closed-ended formats, leaving mathematical proof problems underrepresented.

Automated Evaluation for Generative Outputs. The LLM-as-a-judge approach uses LLMs to score open-ended responses on axes such as correctness, enabling scalable evaluation with reduced human cost (Zheng et al., 2023b; Li et al., 2024). Recent work has examined judge reliability for more challenging tasks (Tan et al., 2025; Frick et al., 2024). For mathematical proofs, automated evaluation remains less explored. Existing work is often highly specialized, such as IneqMath (Sheng et al., 2025) for inequality proofs, or lacks fine-grained evaluation, e.g, the dataset from Dekoninck et al. (2025) provides only binary correctness annotations. Consequently, many studies still rely on manual evaluation to assess the quality of model-generated proofs (Petrov et al., 2025; Huang & Yang, 2025).

Formal Proof Generation. A complementary line of work leverages interactive theorem provers (ITPs), where LLMs generate proofs in formal languages like Lean or Isabelle. The correctness of these proofs can be automatically verified by the ITP, guaranteeing their logical soundness. Recent benchmarks in this area include miniF2F (Zheng et al., 2021), FIMO (Liu et al., 2023), PutnamBench (Tsoukalas et al., 2024), which formalize competition math problems; and LeanWorkbook (Ying et al., 2024), which provides a broad Lean corpus for training and evaluation. While this approach is promising, our work focuses on informal proofs for two key reasons. First, natural language remains the primary medium for mathematical communication in both education and research. Second, the task of automatically translating informal proofs into a formal language, i.e., autoformalization, is itself an unsolved and exceptionally challenging research problem (Gao et al., 2025; Liu et al., 2025).

Summary. While prior work has established the importance of proof-based evaluation and explored the "LLM-as-a-judge" paradigm, a critical gap remains. No existing work has conducted a systematic, empirical study of the evaluator design space for mathematical proofs, nor provided a scalable methodology for creating the fine-grained, rubric-driven annotations necessary for such a study. The key factors that determine evaluator robustness remain largely uninvestigated.

3 METHODS: DEVELOPING AUTOMATED EVALUATORS

Developing an automated evaluator for model-generated mathematical proofs is essential for two reasons: it enables scalable assessment of LLMs, overcoming the limitations of costly and time-consuming human grading, and it provides a reliable reward signal to enhance proof generation capabilities.

In this paper, our goal is to design an evaluator that, given a problem x, a set of reference solutions $\mathcal{S}$, and a model-generated solution s, produces a rating that is an integer between 0 and 7:

$$\psi : (x, s) \mapsto y, y \in \mathbb{Z}, 0 \leq y \leq 7.$$

We will demonstrate the superiority of this scoring granularity over binary grading (correct/incorrect) in §6.

The core of our approach is a marking-scheme-driven methodology. We first generate a detailed, problem-specific marking scheme to provide a consistent and explicit guideline for evaluation. From this, we conduct a systematic study of various key design choices that influence an evaluator's performance. Our study investigates four primary axes of design: (1) the backbone LLM, (2) the contextual information provided (e.g., the marking scheme, reference solutions), (3) the instruction set given to the model, and (4) the overall evaluation workflow (e.g., single-pass, ensemble, or staged). We quantitatively compare these different approaches in §5 to determine an optimal design.

3.1 AUTOMATED MARKING SCHEME GENERATION

A central challenge in fine-grained evaluation is maintaining consistency across different solutions. To address this, our methodology begins by creating a detailed, problem-specific marking scheme for each problem. This scheme serves as a generator-agnostic, structured rubric that can guide both our automated evaluators and human experts.

The marking scheme is produced by an LLM $\mathcal{M}_{\text{MS}}$. Given the problem x and reference solutions $\mathcal{S}$, $\mathcal{M}_{\text{MS}}$ is prompted to output (a) a list of conditions under which scores are awarded or deducted, and (2) a list of trivial cases that should not be awarded any points. See §A.6 for an example. While human-written rubrics are the gold standard, automated generation offers superior scalability and a systematic way to recognize key steps from provided reference solutions.

3.2 A SYSTEMATIC STUDY OF EVALUATOR DESIGNS

Using the pre-generated marking schemes, we then conduct a systematic study to identify the most effective designs for the final evaluator, ψ. We explored a range of designs, from simple single-pass methods to more complex ensemble and staged workflows.

3.2.1 SINGLE-PASS METHODS

A single-pass evaluator prompts a backbone model $\mathcal{M}$ (which may or may not be the same as $\mathcal{M}_{\text{MS}}$) to grade a solution s in one step. We analyze single-pass evaluator along three dimensions: the backbone LM (details in §5.2), the context provided and the instructions given.

Context. We consider four different context configurations to measure the impact of in-context information on performance. These include providing the evaluator with both the pre-generated marking scheme and the reference solution(s) (REF+MS); providing only the marking scheme (MS); providing only the reference solution(s) (REF); and a zero-shot baseline where the evaluator receives only basic grading instructions without any problem-specific context (NONE).

Instruction. For the most informative context setting, REF+MS, we compared three types of instructions to understand how guidance on using the context affects performance. These include NORM (Normal), a flexible instruction that directs the model to follow the marking scheme but allows it to map valid alternative approaches to equivalent checkpoints; STRICT, a rigid instruction that requires the model to adhere strictly to the provided marking scheme and penalize any deviation; and BASIC, an instruction with minimal guidance on how to use the provided materials.

3.2.2 ENSEMBLE-BASED METHODS

To improve the robustness and reduce the variance of single-pass evaluators, we explored a simple ensembling technique. This non-adaptive procedure involves running the same evaluator independently multiple times and combining the individual ratings with an aggregation operator, such as the mean or median, to produce a more stable final score.

3.2.3 STAGED EVALUATION METHODS

Finally, we investigated a staged workflow that decomposes the complex task of evaluation into a fixed sequence of simpler steps. Specifically, we implemented a **Binary & Errors $\rightarrow$ Fine-Grained** workflow. In this two-step process, a first pass predicts the binary correctness of the proof and identifies a list of key errors. A second pass then uses these discrete signals as input to generate a more calibrated and accurate 0–7 score. We found this method particularly effective for improving the performance of weaker backbone models.

4 PROOFBENCH: DATASET WITH MATH PROOF PROBLEMS AND EXPERT RATINGS

We assess proof evaluators for *human alignment* on a 0–7 scale, which requires fine-grained expert annotations on model-generated proofs across diverse sources and years. Existing resources are limited in scale or scoring granularity (Petrov et al., 2025; Dekoninck et al., 2025). We therefore construct our own dataset, PROOFBENCH, consisting of 131 proof problems from six major math competitions with corresponding model solutions by the state-of-the-art reasoning models: o3, Gemini 2.5 Pro, and DeepSeek-R1. This section describes the data collection, annotation process, and statistics; §5 then describes how we use it to evaluate proof evaluators.

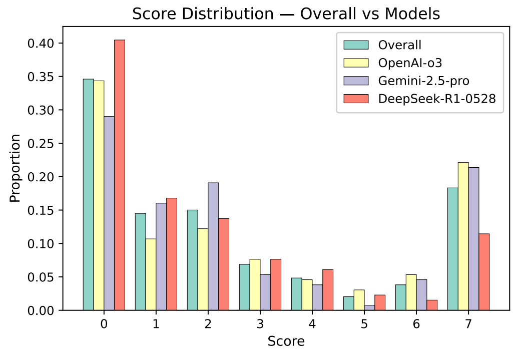
4.1 PROBLEM COLLECTION

We collected 131 problems from the official websites of prestigious mathematics competitions, including the APMO, EGMO, IMO, PUTNAM, USA TST, and USAMO, spanning the years 2022–2025. Sourcing directly from official materials ensures the reliability of problem statements and ground-truth solutions, avoiding transcription errors common in secondary sources. Problems were parsed from official PDFs and normalized, and all available human solutions were included. The final collected data consists of a set of `(problem, reference solution(s), metadata)` triples, where metadata includes the source competition and year.

4.2 EXPERT GRADING PIPELINE

We created our dataset using a rigorous two-stage expert grading pipeline, designed to produce high-quality, marking-scheme-driven evaluation data. The entire process was conducted by two experts with Putnam-level competition experience, supported by a unified web interface.

Stage 1: Marking Scheme Finalization. Our pipeline first generates a detailed marking scheme for each problem given reference solutions. This is achieved using a frozen generator based on



Contest	Count	%
PUTNAM	34	26.0
EGMO	23	17.6
USAMO	23	17.6
IMO	19	14.5
APMO	18	13.7
TST	14	10.7
Total	**131**	**100.0**

Figure 1: Dataset summary: scores by bin (left) and problems per source (right).

gemini-2.5-pro, which was developed through a rigorous process of model selection and iterative prompt refinement based on expert feedback. The full details of the pilot study and refinement process for the rubric generator are available in §A.3.

Stage 2: Marking-Scheme-Guided Solution Grading. With a problem-specific marking scheme, an expert annotator scores a given model-generated proof. Crucially, the experts were instructed to treat the marking scheme as a detailed reference for the expected solution path, rather than a rigid checklist. This is particularly important for fairly evaluating proofs that employed a novel method different from the provided ground-truth solutions, ensuring that valid alternative reasoning paths were credited appropriately. The experts will assign an overall score on a 0–7 scale. Before large-scale annotation, our experts underwent a calibration phase to align their judgments and finalize rules for handling edge cases, a critical step for minimizing inconsistencies.

4.3 Dataset Statistics

We applied our annotation pipeline to the collected problems and for each problem, we generated one proof from three state-of-the-art reasoning models: OpenAI o3, gemini-2.5-pro, and DeepSeek-R1-0528, which span both proprietary and open-source families. For each problem in our collection, we generated one proof from each model using a standardized prompt (§A.7) that asks for a complete, self-contained proof suitable for expert grading.

This process yielded 391 expert-annotated, rubric-guided evaluations. Each sample in the final dataset is a comprehensive entry containing: (i) the problem statement, its official reference solutions, and associated metadata; (ii) the problem-specific marking scheme used for grading, (iii) the LLM-generated proof; and (iv) the final expert evaluation, which provides a fine-grained assessment including an overall expert score (0–7), a holistic overall comment, and a list of specific expert-identified errors. Figure 1 reports the distribution of these final scores across the different models, providing a detailed view of the current landscape in proof-generation capabilities.

5 Evaluation Results

Using ProofBench, we systematically study evaluator design. We first define the assessment metrics (§5.1); then analyze *single-pass* evaluators by ablating the model backbone, context components, and instruction sets (§5.2); next evaluate *ensemble* methods (§5.3) and a *staged* evaluator (§5.4).

Overall, we demonstrate that (i) a strong reasoning backbone with informative context yields substantial gains, (ii) simple ensembling further improves accuracy and robustness, and (iii) a *staged* evaluation pipeline can refine weaker model backbones.

Model	Context	RMSE ↓	MAE ↓	WTA$_{\leq 1}$ (%) ↑	kendall-τ ↑	Bias → 0
O3	REF+MS	**1.194**	**0.899**	**78.4%**	**0.531**	**−0.001**
	MS	1.355	1.024	74.0%	0.486	−0.365
	REF	1.530	1.291	65.9%	0.501	0.475
	NONE	1.861	1.648	50.6%	0.457	0.920
GEMINI	REF+MS	1.612	1.279	63.9%	**0.564**	0.625
	MS	**1.399**	**1.060**	**71.5%**	0.509	**0.113**
	REF	2.135	1.874	40.7%	0.451	1.274
	NONE	2.379	2.093	37.4%	0.328	1.513
O4-MINI	REF+MS	1.723	1.294	69.0%	0.493	0.744
	MS	**1.559**	**1.167**	**71.0%**	**0.502**	**0.286**
	REF	1.841	1.487	62.1%	0.449	0.948
	NONE	2.269	1.902	49.6%	0.419	1.589
GPT-5	REF+MS	**1.258**	0.983	74.8%	**0.553**	0.296
	MS	1.266	**0.953**	**76.6%**	0.548	**−0.177**
	REF	1.573	1.368	57.5%	0.515	0.793
	NONE	1.882	1.678	47.1%	0.429	1.039
R1	REF+MS	1.627	1.263	68.8%	0.458	0.729
	MS	**1.581**	**1.203**	**70.9%**	**0.468**	**0.419**
	REF	3.135	2.683	31.6%	0.355	2.459
	NONE	3.279	2.854	33.5%	0.122	2.648
GPT-4O	REF+MS	2.594	2.201	39.1%	**0.469**	1.802
	MS	**2.233**	**1.822**	**50.4%**	0.363	**1.018**
	REF	2.719	2.363	36.4%	0.312	1.856
	NONE	3.467	3.062	30.8%	0.162	2.653

Table 1: **Performance comparison of six language models on 0-7 scale evaluation tasks under different context designs.**

5.1 EVALUATION METRICS

We evaluate on the 0–7 scale using per-problem metrics. All problems have the same number of responses (n). Let P be the number of problems. For each problem $p = 1, \ldots, P$ with expert scores y_{pi} and evaluator outputs $\hat{y}_{pi}$, we compute

$$\mathrm{MAE}_p = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_{pi} - y_{pi}\right|, \quad \mathrm{RMSE}_p = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_{pi} - y_{pi}\right)^2},$$

$$\mathrm{Bias}_p = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_{pi} - y_{pi}\right), \quad \mathrm{WTA}_p(\leq 1) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{\,|\hat{y}_{pi} - y_{pi}| \leq 1\,\}.$$

For ranking agreement within a problem, we use Kendall's τ_b (ties-adjusted); detailed definitions are provided in Appendix A.4. We report macro-averages over problems; aggregation formulas appear in Appendix A.4.

5.2 SINGLE-PASS EVALUATORS

We find that single-pass evaluator performance is largely determined by the backbone model's reasoning capabilities, the provided context components, and the task-specific instructions.

5.2.1 CONTEXT AND MODEL CHOICE

We compare six models that span different model families, size and reasoning capabilities: O3, GPT-5 (GPT-5-Thinking), GEMINI (gemini-2.5-pro), O4-MINI, R1 (DeepSeek-R1-0528), and GPT-4O. For each model, we test four context configurations: providing both a reference solution and a marking scheme (REF+MS), only the marking scheme (MS), only the reference solution (REF), or no context to establish a zero-shot baseline (NONE). The results are presented in Table 1.

Model	Instruction	RMSE ↓	MAE ↓	$\text{WTA}_{\leq 1}$ (%) ↑	kendall-τ ↑	Bias → 0
O3	NORM	$\mathbf{1.194}$	$\mathbf{0.899}$	$\mathbf{78.4}$	$\mathbf{0.531}$	$\mathbf{-0.001}$
	STRICT	1.341	1.032	74.6	0.473	−0.296
	BASIC	1.271	0.981	74.8	0.514	0.172
O4-MINI	NORM	1.723	1.294	69.0	$\mathbf{0.493}$	0.744
	STRICT	$\mathbf{1.644}$	$\mathbf{1.202}$	$\mathbf{70.5}$	0.443	$\mathbf{0.401}$
	BASIC	1.735	1.291	69.2	0.442	0.714
GEMINI	NORM	1.612	1.279	63.9	$\mathbf{0.564}$	0.625
	STRICT	$\mathbf{1.506}$	$\mathbf{1.174}$	$\mathbf{68.7}$	0.498	$\mathbf{0.294}$
	BASIC	1.690	1.355	62.8	0.481	0.816

Table 2: **Instruction ablation under REF+MS, 0–7 scale.**

Model capability is a primary determinant of performance. We observe a clear hierarchy among models. O3 was the top performer, achieving the lowest errors (RMSE 1.194, MAE 0.899) and the highest agreement with human experts ($\text{WTA}_{\leq 1}$ of 78.4%). GPT-5 follows closely, showing comparable strength. GEMINI and O4-MINI are middle-tier models, while R1 and GPT-4O lag significantly. This indicates that strong mathematical reasoning capabilities are essential for reliable evaluation.

Informative context is critical for all models. The largest gains arise from providing contextual information. Evaluators without any context (NONE) have the weakest performance, with RMSE often roughly double compared to other settings with context. For example, for O3, RMSE decreases from 1.861 (NONE) to 1.194 when both a marking scheme and a reference solution are available (REF+MS).

The marking scheme is the most impactful context component. Comparing different context settings, marking scheme (MS) consistently provides a higher refinement than reference solution (REF). For example, for O3, only including marking scheme gives an RMSE of 1.355, which is lower than only including the reference solution (1.530). This suggests that having a structured rubric offers a more direct and reliable scaffold for scoring than a single, monolithic example. For our strongest model, O3, including both context types (REF+MS) yields the best results.

5.2.2 INSTRUCTION SET UNDER REF+MS

We next investigate how the instruction set impacts evaluator performance under the REF+MS context, where both a reference solution and a marking scheme are provided. We compare three instruction types: BASIC (minimal guidance), STRICT (requiring rigid adherence to the marking scheme), and NORM (normal version, allowing flexibility for valid alternative solutions). See the instruction prompts in Appendix A.7.

As Table 2 shows, the optimal instruction depends on the backbone. For the strong O3, the flexible NORM prompt performs best—lowest RMSE (1.194) and near-zero bias (−0.001)—likely because its reasoning can adaptively map diverse solution paths to rubric checkpoints, whereas STRICT over-constrains. For O4-MINI and GEMINI, STRICT beats NORM and BASIC, suggesting tighter rules prevent over-crediting plausible but flawed steps. Overall, STRICT improves absolute calibration (RMSE/MAE/Bias), while NORM often slightly leads on ranking fidelity (Kendall-τ), revealing a calibration–ranking trade-off and the need to match instructions to model capability.

5.3 ENSEMBLE-BASED EVALUATORS

While a strong backbone LM and context are crucial, single-pass evaluators can still exhibit performance variance during different runs. To improve robustness, we investigate the effectiveness of ensembling. We evaluate our best single-pass evaluator, O3 under REF+MS, by running it five times and aggregating the 0–7 scores.

As shown in Table 3, ensembling not only stabilizes performance but also improves overall accuracy, surpassing even the best-performing individual run. The average RMSE of a single run is 1.186 ±

Model	Run	RMSE ↓	MAE ↓	$\text{WTA}_{\leq 1}$(%)↑	kendall-τ ↑	Bias → 0
O3	Single (mean±std; n=5)	1.186 (±0.042)	0.924 (±0.028)	77.0 (±1.000)	0.530 (±0.024)	0.030 (±0.018)
	Best single	1.142	0.885	78.8	0.574	0.020
	MEAN	**1.093**	0.884	71.5	**0.600**	0.030
	MEDIAN	1.098	**0.861**	**79.6**	0.571	0.024
	MAJORITY	1.099	**0.861**	77.4	0.553	0.020

Table 3: **Ensembling over multiple runs boosts performance and reduces variance for the O3 evaluator.** We compare five individual runs against three aggregation strategies. Both mean and median aggregation achieve a lower RMSE than the best single run. Mean aggregation is optimal for RMSE and ranking correlation (Kendall-τ), while median aggregation excels on MAE and $\text{WTA}_{\leq 1}$.

Model	Design	RMSE ↓	MAE ↓	$\text{WTA}_{\leq 1}$ (%) ↑	kendall-τ ↑	Bias → 0
O3	Staged	1.236	0.945	76.8	0.507	−0.055
	Single-pass REF+MS	1.194	0.899	78.4	0.531	−0.001
O4-MINI	Staged	1.512	1.131	68.4	0.535	0.363
	Single-pass REF+MS	1.723	1.294	69.0	0.493	0.744
O3+O4-MINI	Staged	1.382	1.027	73.0	0.490	−0.078

Table 4: Staged vs. single-pass evaluator for O3 and O4-MINI

0.042, with the best of five achieving an RMSE of 1.142. By simply averaging the scores across the five runs (MEAN), we reduce the RMSE further to 1.093 and achieve the highest ranking correlation (Kendall-τ of 0.600).

While mean aggregation is optimal for minimizing squared error, taking the median score (MEDIAN) is most effective for minimizing mean absolute error (MAE of 0.861) and maximizing the agreement with human experts ($\text{WTA}_{\leq 1}$ of 79.6%). Overall, ensembling produces a more reliable evaluator than any single run, making it a simple and highly effective technique for refining evaluator.

5.4 STAGED EVALUATORS

We also investigate a staged, **Binary**→ **Fine-Grained** workflow, where a first pass identifies binary correctness and error list to guide a second pass in generating a calibrated 0–7 score.

As shown in Table 4, this workflow's effectiveness is highly model-dependent. For a mid-tier model like O4-MINI, the decomposition is highly beneficial, reducing RMSE by 12% (to 1.512) and halving its positive bias. The initial binary check acts as a scaffold, preventing the model from over-crediting flawed solutions.

Conversely, for the strongest model, O3, the staged pipeline is counterproductive, degrading performance across all metrics (e.g., RMSE increases from 1.194 to 1.236). For such a capable model, the decomposition imposes unnecessary constraints and risks error propagation. A hybrid O3+O4-MINI pipeline also failed to surpass the single-pass O3 evaluator.

6 FEASIBILITY OF EVALUATOR AS REWARD MODEL

A good reward model for proof generation should be able to accurately distinguish good and bad responses. To understand if improving evaluator reliability also improves reward model quality, we study the performance of evaluators under best-of-n (BoN) sampling (Appendix A.5).

Experimental Setup. To create a testbed, we used O3 to generate 8 candidate proofs for each of 29 selected problems from 2025, resulting in 232 unique proofs. These candidates were then scored by human experts using the pipeline described in§4. For this analysis, we test a selection of the fine-grained, single-pass evaluators studied previously. We also include binary evaluators as

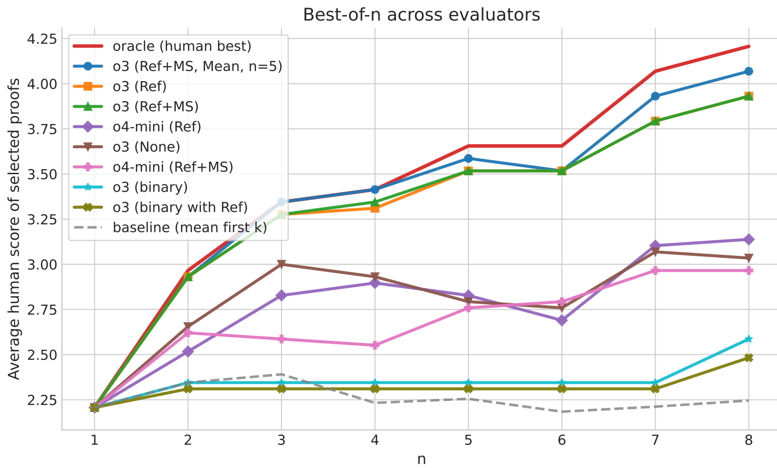


Figure 2: **Best-of-n with different evaluators.** Average best-of-n score over 29 problems for o3 (Generator) as n increases from 1 to 8. o3 (Ref + MS, Mean, n=5) closely track the Human-Oracle@n curve, while the binary evaluators perform much worse.

a simple baseline. These are models prompted to classify a proof with a binary label (Correct or Incorrect) rather than assigning a score on the 0–7 scale, allowing us to test the value of fine-grained feedback. To plot the BoN curve for an evaluator, for each $n \in \{1, \cdots, 8\}$, we select the proof with the highest evaluator-assigned score from the first n candidates, picking the first one to break any ties. The performance at each n is the average human score of these selected proofs across all 29 problems.

Figure 2 shows that offline accuracy (e.g., RMSE) predicts BoN utility. Our best fine-grained model (an ensemble of five o3 runs with REF+MS) demonstrates this by closely tracking the human-oracle curve, consistently selecting higher-quality proofs as n increases. At n=8, our best evaluator achieves an average score of 4.05. This substantially improves upon the binary baseline's score of 2.59 and comes within 0.16 points of the human oracle (4.21).

In contrast, the binary evaluator fails to improve, performing only slightly better than the average of all candidates. This failure highlights a critical limitation: by collapsing all "correct" proofs into a single category, the model loses the ability to rank solutions. When multiple correct candidates exist, it cannot distinguish an adequate proof (e.g., a 5/7) from an excellent one (7/7). Fine-grained scoring preserves relative ordering, making it essential for effective reward models in mathematical reasoning.

7 Conclusion

In this work, we addressed the critical challenge of reliably evaluating natural language mathematical proofs. We present PROOFBENCH, the first comprehensive annotated dataset of fine-grained proof grading that spans multiple contests and years. Using PROOFBENCH, we systematically explore the evaluator design space and find that quality depends on a strong model backbone, informative context, and problem-specific marking schemes, and that ensembling further improves accuracy and robustness. Our analysis yields PROOFGRADER, the strongest evaluator considering all design choice. As a downstream test, using the evaluator as a reward signal for best-of-N selection closely tracks human-oracle performance, demonstrating practical utility for training and selection. We expect PROOFBENCH and PROOFGRADER, together with our analysis, to provide strong foundations for future work on open-ended mathematical reasoning.

Ethics Statement

Our work adheres to the ICLR Code of Ethics. The primary ethical considerations for this research involve the creation and use of our dataset, ProofBench, and the potential for biases in our LLM-based evaluator. The dataset was constructed using problems from publicly accessible mathematics competitions. The expert annotations, which form the core of our benchmark, were provided by qualified individuals who were fairly compensated for their time and expertise. To protect their privacy, all personally identifiable information has been removed from the released data.

Reproducibility Statement

We are committed to ensuring the reproducibility of our research. To this end, we will publicly release all code, data, and supplementary materials upon publication. The ProofBench, including all problems, LLM-generated solutions, expert-annotated scores, and problem-specific marking schemes, will be made available. Our source code will include scripts to reproduce the experiments presented in this paper, including the implementation of our evaluator, ProofGrader, all prompts used, and the analysis notebooks to generate our figures and tables. Detailed descriptions of our experimental setup, model versions, and hyperparameters are provided in the Appendix.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2025. URL https://arxiv.org/abs/2505.23281.

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

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

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.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios N. Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. How to evaluate reward models for rlhf, 2024. URL `https://arxiv.org/abs/2410.14872`.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024. URL `https://arxiv.org/abs/2410.07985`.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=Se6MgCtRhz`.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024. URL `https://arxiv.org/abs/2402.14008`.

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

Yichen Huang and Lin F. Yang. Gemini 2.5 pro capable of winning gold at imo 2025, 2025. URL `https://arxiv.org/abs/2507.15855`.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline, 2024. URL `https://arxiv.org/abs/2406.11939`.

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, et al. Fimo: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*, 2023.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=hUb2At2DsQ`.

Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating llms on 2025 usa math olympiad. *arXiv preprint arXiv:2503.21934*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL `https://arxiv.org/abs/2402.03300`.

Jiayi Sheng, Luna Lyu, Jikai Jin, Tony Xia, Alex Gu, James Zou, and Pan Lu. Solving inequality proofs with large language models, 2025. URL `https://arxiv.org/abs/2506.07927`.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges, 2025. URL `https://arxiv.org/abs/2410.12784`.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *Advances in Neural Information Processing Systems*, 37: 11545–11569, 2024.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025. URL https://arxiv.org/abs/2504.20571.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Advances in Neural Information Processing Systems*, 37:105848–105863, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Albert S. Yue, Lovish Madaan, Ted Moskovitz, DJ Strouse, and Aaditya K. Singh. Harp: A challenging human-annotated math reasoning benchmark, 2024. URL https://arxiv.org/abs/2412.08819.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023a. URL https://openreview.net/forum?id=uccHPGDlao.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023b. URL https://arxiv.org/abs/2306.05685.

A Appendix

A.1 The Use of Large Language Models (LLMs)

In preparing this submission, we used LLMs as a general-purpose writing assistant. The human authors were responsible for the entire research process, including the initial ideation, experimental design, implementation, and data analysis.

During writing, authors first creating a complete draft of each section, which included all core ideas and technical details. The LLM was then used to revise this draft for clarity, conciseness, and grammatical correctness. Its role was strictly limited to improving the language and flow of the text.

All scientific contributions and claims presented in this paper originate from the authors, who take full responsibility for the final content.

A.2 Data Information

The data information for the collected problems are available in Table 5

Contest	2022	2023	2024	2025
APMO	5	5	3	5
EGMO	6	5	6	6
IMO	4	4	5	6
PUTNAM	12	10	12	–
TST	–	4	6	4
USAMO	6	5	6	6
Total per year	**33**	**33**	**38**	**27**

Table 5: Core problem counts by contest and year.

A.3 Annotation Pipeline Details

Stage 1: Marking Scheme Finalization. *Pilot.* We compare rubric generators from two model families, each *with* and *without* in-context examples, using an initial prompt distilled from authoritative grading materials.

Quality rating. For each problem–solution pair, annotators independently rate the generated marking scheme on a 0–3 scale (0 = invalid; 3 = high-fidelity), then discuss to consensus.

Selection and refinement. We select the best configuration (gemini-2.5-pro, no in-context example), iteratively refine the prompt based on annotator feedback, and re-evaluate on additional problems until agreement stabilizes. We then **freeze** the marking scheme generator for subsequent use.

Stage 2: Solution annotation. *Pilot calibration.* Using the frozen rubric generator, we produce per-problem marking schemes and calibrate the scoring protocol. Two experts will both annotate 36 problems, 3 responses each. They will discuss disagreement to reach consensus and adjust their grading protocol.

Scale and quality control. We apply the calibrated protocol to 131 problems with 3 solutions each, yielding 393 rubric-guided annotations. We double-score 20% of items, run periodic drift checks, and adjudicate all flagged disagreements.

A.4 Additional details for evaluation metrics

Within-problem ranking agreement. For problem p, consider all pairs (i, j) with $i < j$. Define $\Delta_{ij}^{\text{exp}} = y_{pi} - y_{pj}$ and $\Delta_{ij}^{\text{eval}} = \hat{y}_{pi} - \hat{y}_{pj}$. Let C and D be the numbers of concordant and discordant pairs, respectively, and let

$$T_{\text{exp}} = \#\{(i,j) : \Delta_{ij}^{\text{exp}} = 0\}, \qquad T_{\text{eval}} = \#\{(i,j) : \Delta_{ij}^{\text{eval}} = 0\}.$$

Kendall's τ_b for problem p is

$$\tau_b(p) = \frac{C - D}{\sqrt{(C + D + T_{\text{exp}})\,(C + D + T_{\text{eval}})}},$$

and is undefined if the denominator is zero (we omit such p from aggregation).

Macro averaging. We macro-average per-problem metrics across problems. Writing $\mathcal{P}$ for the set of problems with defined τ_b and $P' = |\mathcal{P}|$:

$$\overline{\text{MAE}} = \frac{1}{P}\sum_{p=1}^{P} \text{MAE}_p, \quad \overline{\text{RMSE}} = \frac{1}{P}\sum_{p=1}^{P} \text{RMSE}_p, \quad \overline{\text{Bias}} = \frac{1}{P}\sum_{p=1}^{P} \text{Bias}_p, \quad \overline{\text{WTA}}(\leq 1) = \frac{1}{P}\sum_{p=1}^{P} \text{WTA}_p(\leq 1),$$

$$\overline{\tau_b} = \frac{1}{P'} \sum_{p \in \mathcal{P}} \tau_b(p).$$

(As noted in the main text, all problems have the same response count n.)

A.5 Best-of-n

A good reward model for proof generation should be able to accurately distinguish good and bad responses. To understand if improving evaluator reliability also improves reward model quality, we study the performance of evaluators under best-of-n (BoN) sampling, which evaluates a reward model by its ability to select the best response under the same budgeted sampling that training uses: n candidates are sampled from a fixed policy π, the best solution is chosen as $\hat{y} = \arg\max_i \mathrm{RM}(y_i)$, and the true quality is measured $R(\hat{y})$. Unlike global correlations or calibration metrics, BoN is conditioned on the policy of the generator and the inference budget, directly reflecting operations that drives post-training (from sampling to scoring to selecting the best rollout). It reveals reward over-optimization, demonstrates robustness under affine score drift, and yields a practical utility curve as n varies (typically 8). In general, higher BoN means the RM more reliably upgrades the data the learner trains on—predicting real downstream gains.

Best-of-n is also well-explored in other works as a metric for evaluating reward model and evaluator robustness (Frick et al., 2024; Tan et al., 2025).

A.6 Marking Scheme Example

Marking Scheme for APMO-2025-2

Here is the grader-friendly rubric for the provided problem and solution.
Checkpoints (max 7 pts total)
* **1. Path parameterization [1 pt, additive]** * States and justifies that for the point (x_n, y_n) visited at time n, we have $x_n + y_n = n$.
* **2. Sequence of values [2 pts, additive]** * Proves that the sequence of written numbers must be $z_n = n$ for all $n \geq 0$. * *A complete proof requires showing that the sequence z_n is non-decreasing and then using the condition that the set of values $\{z_n\}_{n\geq 0}$ is exactly the set of non-negative integers.*
* **3. Necessity of $\alpha + \beta = 2$ [2 pts, additive]**
* Correctly uses the constraint $|x_n - y_n| < 2025$ to derive a bound on the term $x_n\alpha + y_n\beta$. The bound must show that $x_n\alpha + y_n\beta$ is in an interval of constant width centered at $n\frac{\alpha+\beta}{2}$. For example, $|(x_n\alpha + y_n\beta) - n\frac{\alpha+\beta}{2}| < C_1$ for some constant C_1. **[1 pt]**
* Combines the bound above with $z_n = n$ (and the property of the floor function, e.g., $v-1 < \lfloor v \rfloor \leq v$) to obtain an inequality of the form $|n - n\frac{\alpha+\beta}{2}| < C_2$ for some constant C_2, and concludes from this that $\alpha + \beta = 2$ is necessary (e.g., via an unboundedness argument for large n). **[An additional 1 pt]**
* **4. Sufficiency of $\alpha + \beta = 2$ [2 pts, additive]**
* Proposes a concrete path for all n and verifies it is valid (i.e., starts at $(0, 0)$, consists of unit steps right or up, and satisfies $|x_n - y_n| < 2025$). For example, $(x_n, y_n) = (\lceil n/2 \rceil, \lfloor n/2 \rfloor)$. **[1 pt]**
* Verifies that for *any* pair of positive reals (α, β) with $\alpha + \beta = 2$, a path can be chosen such that $\lfloor x_n\alpha + y_n\beta \rfloor = n$ for all n. This requires checking both even and odd n and showing that a suitable path exists regardless of whether $\alpha \geq \beta$ or $\alpha < \beta$. **[An additional 1 pt]**
* **Total (max 7)**
Zero-credit items
* Stating the answer $\alpha + \beta = 2$ without any valid justification.
* Only checking a few small values of n (e.g., $n = 1, 2$).
* Asserting $z_n = n$ without providing any argument (e.g., without showing z_n is non-decreasing).
* Any reasoning that ignores or mishandles the floor function $\lfloor\cdot\rfloor$.

* Stating the bounds on x_n and y_n from the $|x - y| < 2025$ condition, but not using them to constrain z_n.
Deductions
* **D-1: Minor error in bounds:** A minor algebraic or logical error in deriving the bounds for the necessity part that does not invalidate the overall structure of the argument (e.g., omitting the '+1' from the floor function). **(–1)**
* **D-2: Incomplete sufficiency proof:** The proof of sufficiency constructs a path and verifies it works only for a proper subset of solution pairs (e.g., only for $\alpha \geq \beta$, or only for $(\alpha, \beta) = (1, 1)$), without justifying that a construction is possible for all pairs with $\alpha + \beta = 2$. **(–1)**
* **D-3: Contradictory statements:** The submission contains logically contradictory claims (e.g., asserting that the path must be unique and also that multiple paths are possible). **(cap at 5/7)**

A.7 Prompts

Below we list all prompts used in our evaluation. Each prompt is shown verbatim.

With Reference Solution and Marking Scheme

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.
- **Reference Solution**: A correct solution or proof provided for reference. This is **not necessarily the only valid solution**. If the problem requires a final numeric or algebraic answer, this section contains the correct answer, which should be the only accepted final answer (though alternative reasoning paths are valid).
- **Marking Scheme**: A problem-specific grading rubric (0–7 scale) with checkpoints, zero-credit items, and deductions. **Treat this scheme as advisory guidance, not a script.** Use it to anchor scoring, but **do not require** the proof to follow the same order, lemmas, or technique if its reasoning is mathematically sound.
- **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 stated).
3. **Advisory mapping to the marking scheme** (checkpoints/deductions), allowing different orders and techniques.
4. **Reference solution** as an anchor for sufficiency, not exclusivity.

Alternative-approach policy:

- If the proof uses a different but valid method, **map its steps to equivalent rubric checkpoints** (same logical role) and award points accordingly.
- **Do not penalize** solely for re-ordering steps, using different lemmas, or giving a correct shortcut, **unless** the problem forbids it.
- Apply zero-credit items/deductions **only when the underlying issue actually occurs** in the given proof's approach; **do not auto-penalize** for omitting a rubric step that is unnecessary under the alternative method.

- Avoid double-counting mutually exclusive items; if two items solve the same logical gap, **award the larger only**.
- If the final numeric/algebraic answer is wrong where uniqueness is required, award only partial credit justified by correct intermediate reasoning.

Rigor and evidence:

- Award credit for intermediate claims **only if adequately justified** within the proof (not merely asserted).
- If a step is plausible but under-justified, award **conservative partial credit** and note what is missing.

What to produce:

- Identify logical errors, incorrect steps, or unclear reasoning.
- Give a **score between 0 and 7** with a **detailed assessment**.
- **Within the assessment text**, show clearly how the score was derived:
  - Which rubric checkpoints (or their **mapped equivalents**) were earned and the points you awarded.
  - Any zero-credit items or deductions you applied (and why).
  - How these add up to the final integer score in [0–7].

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- `<score>` must be an integer in [0, 7].
- `<assessment>` must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present. Include the scoring breakdown **in prose** here (earned checkpoints or mapped equivalents, deductions, and subtotal → final score).
- `<errors>` must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the main
idea, but has some unclear reasoning and minor
mistakes...</assessment>
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Problem Statement
{problem}

Reference Solution
{human_solution}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}

Basic Evaluation Template

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.
- **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.

- Identify logical errors, incorrect steps, or unclear reasoning.
- Give an **integer** score between 0 and 7 with a brief overall assessment.

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- `<score>` must be an integer in [0, 7].
- `<assessment>` must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present.
- `<errors>` must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the main
idea, but has some unclear reasoning and minor
mistakes...</assessment>
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Scoring Guidelines (0–7 scale)

- **0**: Completely incorrect; proof is irrelevant, nonsensical, or shows no understanding.
- **1–2**: Very poor; major logical flaws, does not solve the problem, but may contain fragments of relevant reasoning.
- **3–4**: Partial progress; captures some correct reasoning or key ideas, but has significant logical errors, missing steps, or incomplete arguments that make the proof invalid overall.
- **5–6**: Largely correct; the proof is overall valid and reaches the correct conclusion. Contains only **minor issues** (e.g., small calculation mistakes, notation slips, or slightly unclear wording) that do not undermine correctness.
- **7**: Fully correct; the proof is complete, logically sound, and clearly presented with no substantive errors.

Problem Statement
{problem}

Proof Solution
{solution}

With Reference Solution

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.
- **Reference Solution**: A correct solution or proof provided for reference. This is **not necessarily the only valid solution**. If the problem requires a final numeric or algebraic answer, this section contains the correct answer, which should be the only accepted final answer (though alternative reasoning paths are valid).
- **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.

- Compare the proof against the reference solution where relevant.
- Identify logical errors, incorrect steps, or unclear reasoning.
- Give a score between 0 and 7 with a brief overall assessment.

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- `<score>` must be an integer in [0, 7].
- `<assessment>` must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present.
- `<errors>` must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the main
idea, but has some unclear reasoning and minor
mistakes...</assessment>
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Scoring Guidelines (0–7 scale)

- **0**: Completely incorrect; proof is irrelevant, nonsensical, or shows no understanding.
- **1–2**: Very poor; major logical flaws, does not solve the problem, but may contain fragments of relevant reasoning.
- **3–4**: Partial progress; captures some correct reasoning or key ideas, but has logical errors, missing steps, or incomplete arguments that make the proof invalid overall.
- **5–6**: Largely correct; the proof is overall valid and reaches the correct conclusion. Contains only **minor issues** (e.g., small calculation mistakes, notation slips, or slightly unclear wording) that do not undermine correctness.
- **7**: Fully correct; the proof is complete, logically sound, and clearly presented with no substantive errors.

Problem Statement
{problem}

Reference Solution
{human_solution}

Proof Solution
{solution}

With Reference Solution and Marking Scheme (Strict)

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.
- **Reference Solution**: A correct solution or proof provided for reference. This is **not necessarily the only valid solution**. If the problem requires a final numeric or algebraic answer, this section contains the correct answer, which should be the only accepted final answer (though alternative reasoning paths are valid).
- **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.

- Compare the proof against the reference solution and the marking scheme.
- Award points according to the marking scheme's checkpoints, zero-credit items, and deductions.
- Identify logical errors, incorrect steps, or unclear reasoning.
- Give a score between 0 and 7 with a brief overall assessment.
- Show clearly how the score was derived:
  - Which checkpoints were earned (with awarded points).
  - Any zero-credit items or deductions applied.
  - How the subtotal leads to the final score (0–7).

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- `<score>` must be an integer in [0, 7].
- `<assessment>` must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present.
- `<errors>` must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the main
idea, but has some unclear reasoning and minor
```

```
mistakes...</assessment>
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Problem Statement
{problem}

Reference Solution
{human_solution}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}

With Reference Solution and Marking Scheme (most basic)

You are an expert grader for math proofs. Judge the proof's mathematical correctness based on the reference solution and the marking scheme, return an integer score between 0 and 7.

INPUTS:

- Problem Statement
- Reference Solution (correct but not exclusive)
- Marking Scheme (0–7) with checkpoints and deductions — use as guidance, not a script
- Proof Solution (from an LLM)

OUTPUT (XML only; no extra text):

```
<score>[integer 0{7]</score>
<assessment>[step-by-step rationale with scoring breakdown in
prose]</assessment>
<errors>[numbered list of specific issues; empty if none]
</errors>
```

Problem Statement
{problem}

Reference Solution
{human_solution}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}

With Reference Solution and Marking Scheme (basic)

You are an expert grader for math proofs.

INPUTS:

- Problem Statement
- Reference Solution (correct but not exclusive)

- Marking Scheme (0–7) with checkpoints and deductions — use as guidance, not a script
- Proof Solution (from an LLM)

TASK: Judge the proof's mathematical correctness. Prefer validity over problem constraints over marking scheme alignment over reference solution. If the proof uses a different valid method, map its steps to equivalent marking scheme checkpoints and award points. If a unique final answer is wrong, give partial credit only for justified intermediate reasoning.

OUTPUT (XML only; no extra text):

```
<score>[integer 0{7]</score>
<assessment>[step-by-step rationale with scoring breakdown in
prose]</assessment>
<errors>[numbered list of specific issues; empty if none]
</errors>
```

Problem Statement
{problem}

Reference Solution
{human_solution}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}

With Marking Scheme (no reference solution)

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.

- Follow the marking scheme exactly: award checkpoints, apply zero-credit items, and apply any deductions/caps as specified.
- Identify logical errors, incorrect steps, or unclear reasoning.
- Give a score between 0 and 7 with a brief overall assessment.
- Show clearly how the score was derived:
  - Which checkpoints were earned (with awarded points).
  - Any zero-credit items or deductions applied.
  - How the subtotal leads to the final score (0–7).

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- `<score>` must be an integer in [0, 7].
- `<assessment>` must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present.
- `<errors>` must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the
main idea, but has some unclear reasoning and minor
mistakes...</assessment>
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Problem Statement
{problem}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}

With Reference Solution and Marking Scheme (more detailed)

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.
- **Reference Solution**: A correct solution or proof provided for reference. This is **not necessarily the only valid solution**. If the problem requires a final numeric or algebraic answer, this section contains the correct answer, which should be the only accepted final answer (though alternative reasoning paths are valid).
- **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.

How to Use the Marking Scheme (mandatory)
1. Checkpoints parsing & awarding

- Treat each checkpoint exactly as written. Respect its tag:

additive : award all applicable items in that bullet/group.
max k : award up to k points from the items in that bullet/group (choose the best-matching ones; do not exceed k).

- If items are nested with "award the larger only", and more than one applies, award only the larger point value.
- If the scheme presents parallel checkpoint chains (alternative legitimate paths), score the single chain or combination that yields the highest valid total without

violating exclusivity or [max k] caps. Do not double-count equivalent steps across mutually exclusive paths.

- If a catch-all checkpoint is provided for a fully correct alternative proof using the same underlying idea, you may award up to its stated maximum only when the student's argument is complete and logically valid for that idea.

2. Zero-credit items
If the proof relies on any listed zero-credit arguments, award 0 for those parts. Do not add points for restatements, conjectures without proof (especially in geometry), or dead-ends.
3. Deductions (apply at most one)

- Identify applicable deductions and apply only the single largest (e.g., -1, -2, or cap at x/7).
- Apply a cap by truncating the post-checkpoint subtotal to x before finalizing the score.
- Never reduce the score below 0. Cosmetic slips (notation, arithmetic, wording) do not trigger deductions unless they break validity.

4. Final answer consistency (when applicable)
If the reference solution gives a definitive final answer, the candidate solution's final answer must be **correct/equivalent**. If not, follow the marking scheme's checkpoints/deductions; typically, a wrong final answer prevents awarding the ”conclusion” checkpoint.
5. Arithmetic & bounds

- Checkpoint awards are integers. Subtotal ≤ 7 by construction.
- After applying the single largest deduction/cap, the final score is an integer in [0, 7].

Task
Analyze the proof carefully.

- Compare the proof against the reference solution and the marking scheme.
- Award points according to the marking scheme's checkpoints, zero-credit items, and deductions.
- Identify logical errors, incorrect steps, or unclear reasoning.
- Give a score between 0 and 7 with a brief overall assessment.
- Show clearly how the score was derived:
  - Which checkpoints were earned (with awarded points).
  - Any zero-credit items or deductions applied.
  - How the subtotal leads to the final score (0–7).

Output Format
Respond with **only** well-formed XML using the structure below. Do not include any extra text or Markdown.
Requirements:

- <score> must be an integer in [0, 7].
- <assessment> must be a **detailed analysis** that explains your reasoning step-by-step and provides a clear **rationale for the score**. Reference specific claims/lines if present.
- <errors> must be a list of specific issues (empty if score = 7).

Example output:

```
<score>0</score>
<assessment>The proof shows a good understanding of the main
idea, but has some unclear reasoning and minor
mistakes...</assessment>
```

```
<errors>
  1. specific error 1,
  2. specific error 2,
  ...
</errors>
```

Problem Statement
{problem}

Reference Solution
{human_solution}

Marking Scheme
{marking_scheme}

Proof Solution
{solution}