Reliable Fine-Grained Evaluation of Natural Language Math Proofs

Wenjie Ma[1*] **Andrei Cojocaru**[1] **Neel Kolhe**[1] **Robin Said Sharif**[1]
Haihan Zhang[3] **Vincent Zhuang**[2†] **Matei Zaharia**[1] **Sewon Min**[1,4]
[1] UC Berkeley [2] Google DeepMind [3] Peking University [4] Allen Institute for AI

Abstract

Recent advances in large language models (LLMs) for math reasoning have largely focused on tasks with easily verifiable final answers; however, generating 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 to model-generated math proofs. We first introduce **PROOFBENCH**, the first expert-annotated dataset of fine-grained proof ratings, spanning 145 problems from six major math competitions and 435 LLM-generated solutions from Gemini-2.5-Pro, o3, and DeepSeek-R1. With PROOFBENCH, we systematically explore the evaluator design space across key axes: the backbone model, input context, instructions and evaluation workflow. Our analysis delivers **PROOFGRADER**, an evaluator that combines a strong reasoning backbone LM, rich context from reference solutions and marking schemes, and a simple ensembling method; it achieves a low Mean Absolute Error (MAE) of 0.926 against expert scores, significantly outperforming naive baselines. Finally, we demonstrate its practical utility in a best-of-n selection task: at $n = 16$, PROOFGRADER achieves an average score of 4.14/7, closing 78% of the gap between a naive binary evaluator (2.48) and the human oracle (4.62), highlighting its potential to advance 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 with verifiable rewards [1, 10, 13–15]. However, these methods do not transfer to proof generation because: (i) many proof problems do not admit an 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 [2, 9]. Because proof-generation tasks constitute a large share of mathematical problem solving in research and education, *reliable proof evaluation* is essential–for faithful capability assessment and as a reward signal for training.

Existing options for this task still fall short. Expert grading, while accurate, is slow and costly. While formal math (e.g., Lean) offers absolute certainty, it remains detached from the natural language used in most human mathematics education and research; furthermore, automatically translating natural-language proofs into formal languages is brittle and remains extremely challenging [4, 7]. Our work therefore focuses on the critical and complementary task of evaluating proofs in their natural representation. While LLM-as-a-judge [2, 6, 11, 16] is promising, its application to math proofs is unsettled, and how design reliable proof evaluators is poorly understood.

*Correspondence: windsey@berkeley.edu
†Served in an advisory capacity only.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

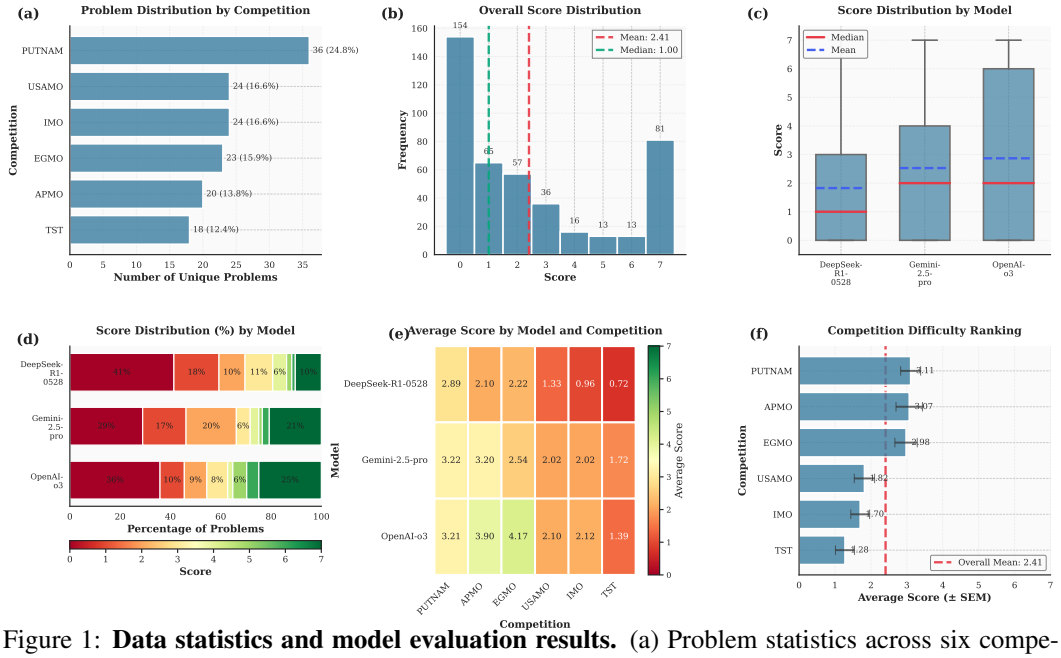

Figure 1: **Data statistics and model evaluation results.** (a) Problem statistics across six competitions. (b) Score distribution showing mean and median. (c) Model performance comparison via box plots. Each box represents the interquartile range (middle 50% of scores). (d) Per-generator score distribution. Stacked horizontal bar chart showing the percentage of problems receiving each score (0-7) for each model. (e) Performance heatmap by generator and competition. (f) Competition difficulty ranking with error bars: TST is the most challenging and Putnam is the easiest.

To address this, we conduct a comprehensive study of proof-evaluator design. To support this analysis, we establish **ProofBench**[3], the first expert-annotated dataset for fine-grained proof evaluation that spans problems from multiple contests and years. It contains 435 LLM-generated solutions to 145 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. Next, 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 ensembling multiple evaluation runs to improve robustness.

Our analysis yields **ProofGrader**[4]: 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 Mean Absolute Error (MAE) of 0.926 against expert scores, significantly outperforming naive baselines. We further validate the evaluators' practical utility in a downstream best-of-n selection task, a standard proxy for assessing its potential as a reward signal [3, 5, 8]. At $n = 16$, ProofGrader achieves an average score of 4.14/7, closing 78% of the performance gap between a naive binary evaluator (2.48) and the human oracle (4.62).

2 ProofBench: Expert-Rated Math Proof Solutions

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. Prior work focuses on a single contest [9] or is limited to binary judgments of proof correctness [2]. We thus construct our own dataset, ProofBench. The 0–7 scale aligns the dataset with the established grading standards of premier mathematics competitions[5] which we source problems from.

[3]The full dataset is available at `huggingface.co/datasets/wenjiema02/ProofBench`.

[4]The code is available at `github.com/euclidgame/proofgrader`.

[5]The official Putnam Competition uses a 0–10 scale. For consistency across competitions, we normalize all expert annotations to a unified 0–7 scale.

Problem Sources and Proof Generation. We collected 145 problems from the official websites of prestigious mathematics competitions, including the APMO, EGMO, IMO, Putnam, USA TST, and USAMO, from year 2022 to 2025. Distribution of contests is shown in Figure 1a. Problems were parsed from official PDFs and normalized, and all available human solutions were included. For each problem, we generate a proof using a standardized prompt (§A.8) that asks for a complete, self-contained proof from three frontier reasoning models (*generators*): OpenAI o3, Gemini-2.5-Pro, and DeepSeek-R1-0528, which span both proprietary and open-source families.

Expert Annotations. Finally, annotation proceeds in two stages: marking scheme generation and model-generated proof grading. All annotations were conducted by a team of five experts with Putnam-level or national Math Olympiad experience, using a carefully designed web interface.

Step 1: Automated Marking Scheme Generation: A central challenge in creating this benchmark was ensuring consistent and scalable expert grading. In our method, the marking scheme is produced by an LLM $\mathcal{M}_{\text{MS}}$ which is prompted to output (i) a list of conditions under which scores are awarded or deducted, and (ii) a list of trivial cases that should not be awarded any points. This generator was developed through a rigorous refinement process where experts also graded the quality of the generated MS, providing feedback that led to the selection of Gemini-2.5-Pro as the final $\mathcal{M}_{\text{MS}}$ (see §A.2 for details). An example of a generated marking scheme is shown in Appendix §A.5. Approximately 85% of our marking schemes were judged to be highly reasonable by experts.

Step 2: Proof Grading: With a problem-specific marking scheme, an expert annotator scores a given model-generated proof on a 0-7 scale. Experts were instructed to treat the marking scheme as a detailed reference for the expected solution path, rather than a rigid checklist. This is important for fairly evaluating proofs that employed a novel method different from the reference solutions. We assign 40% of the solutions to more than one annotator and ensure their agreement is 90% or higher.

Dataset Statistics. In total, PROOFBENCH consists of 145 proof problems and 435 expert-annotated evaluations of proofs. Our analysis reveals several key findings about state-of-the-art reasoning models. First, current models remain far from achieving high scores: even the strongest models obtain scores of 6 or higher on fewer than 30% of problems, as shown in Figure 1d. Second, OpenAI-o3 performs best overall, with a mean score of 2.87, and is closely followed by Gemini-2.5-Pro, according to Figure 1c. Third, performance varies considerably across competitions (Figure 1e,f): all models perform best on PUTNAM, where the mean score is 3.11, but struggle on TST.

3 Systematic Study of Evaluator Designs

Using PROOFBENCH, we then study the key design factors for math proof evaluators that produce fine-grained scores. We show that (i) a strong reasoning backbone with informative context yields substantial gains, (ii) simple ensembling further improves performance.

3.1 Evaluator Designs

We study single-pass evaluators along several dimensions and extend it with ensembling methods.

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, the context provided and the instructions given.

- **Backbone Model Choice.** This refers to the LLM that is prompted to execute the evaluation. We compare five models that span different model families, size and reasoning capabilities: O3, GEMINI (Gemini-2.5-Pro), O4-MINI, R1 (DeepSeek-R1-0528), and GPT-4O.
- **Context.** We consider four different context configurations, including: providing 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 naive baseline where only basic grading instructions without any problem-specific context are provided (NONE).
- **Instruction.** The instruction set refers to the specific prompt that guides the evaluator on how to interpret and apply the provided context information and perform the task. In our study, for the most informative context setting, REF+MS, we compared three types of instructions: NORM

(Normal), a flexible instruction that directs the model to follow the marking scheme but allows it to accept valid alternative approaches; STRICT, a rigid instruction that requires the model to adhere strictly to the provided marking scheme; and BASIC, an instruction with minimal guidance.

Ensemble. We consider a simple ensembling technique, which runs the same evaluator independently multiple times and combines the individual ratings with an aggregation operator.

Evaluation Metrics. We report macro-averaged RMSE, MAE, Bias, and $\text{WTA}_{\leq 1}$, where $\text{WTA}_{\leq 1}$ is the fraction of predictions within ± 1 point of the expert score. Metrics are computed per problem (each with n responses) and then averaged over problems. For within-problem ranking agreement we use Kendall's τ_b (ties-adjusted). Formal definitions and aggregation appear in App. A.3.

3.2 Evaluation Results

Model	Context	RMSE ↓	MAE ↓	$\text{WTA}_{\leq 1}$ (%) ↑	Kendall-τ ↑	Bias ≈	Quality
O3	REF+MS	**1.273** ± 0.16	**0.964** ± 0.12	**76.5** ± 4.3	**0.502**	**-0.008**	■■■
	MS	1.418 ± 0.17	1.069 ± 0.14	72.8 ± 4.3	0.477	-0.381	■■■
	REF	1.575 ± 0.14	1.330 ± 0.12	65.3 ± 4.6	0.481	0.478	■■■
	NONE	1.901 ± 0.15	1.680 ± 0.15	49.5 ± 5.8	0.435	0.924	■■■
GEMINI	REF+MS	1.696 ± 0.17	1.342 ± 0.15	62.7 ± 4.9	**0.529**	0.626	■■■
	MS	**1.502** ± 0.17	**1.142** ± 0.14	**70.0** ± 4.2	0.488	**0.151**	■■■
	REF	2.177 ± 0.14	1.910 ± 0.15	39.9 ± 5.2	0.410	1.285	■■■
	NONE	2.397 ± 0.15	2.107 ± 0.16	36.6 ± 4.9	0.319	1.496	■■■
O4-MINI	REF+MS	1.816 ± 0.22	1.367 ± 0.19	67.6 ± 4.6	0.476	0.762	■■■
	MS	**1.636** ± 0.22	**1.234** ± 0.18	**69.5** ± 4.8	**0.505**	**0.309**	■■■
	REF	1.858 ± 0.19	1.504 ± 0.16	61.3 ± 4.8	0.465	0.950	■■■
	NONE	2.276 ± 0.23	1.914 ± 0.21	49.5 ± 5.3	0.430	1.569	■■■
R1	REF+MS	1.735 ± 0.22	1.357 ± 0.18	66.4 ± 5.2	0.429	0.732	■■■
	MS	**1.682** ± 0.22	**1.298** ± 0.18	**68.5** ± 4.9	**0.450**	**0.422**	■■■
	REF	3.187 ± 0.23	2.736 ± 0.22	30.3 ± 5.0	0.289	2.450	■■■
	NONE	3.273 ± 0.25	2.842 ± 0.25	33.5 ± 4.9	0.102	2.581	■■■
GPT-4O	REF+MS	2.599 ± 0.20	2.197 ± 0.20	39.7 ± 5.0	**0.479**	1.824	■■■
	MS	**2.245** ± 0.21	**1.827** ± 0.19	**50.4** ± 5.3	0.377	**1.001**	■■■
	REF	2.726 ± 0.19	2.371 ± 0.18	36.0 ± 4.9	0.343	1.887	■■■
	NONE	3.402 ± 0.26	3.001 ± 0.26	31.9 ± 4.9	0.208	2.614	■■■

Table 1: **Comparison of five LLMs under different context designs.** Contexts: REF+MS (reference solution + marking scheme), MS (marking scheme only), REF (reference solution only), NONE (no context). Values shown as mean ± 95% confidence interval (CI) margin of error. Best values per model in **bold**; best (■) and worst (■) context highlighted. Arrows: ↓ lower better, ↑ higher better, ≈ closer to zero better. Quality: ■ Excellent, ■ Good, ■ Fair, ■ Poor, ■ Bad.

Effects of backbone model and contextual information. Table 1 shows the results of single-pass evaluators across different backbone models and context settings. First, *the strength of the backbone model* strongly correlates with performance: moving from weaker to stronger models brings better calibrations (O3 leads in nearly all metrics). Second, *contextual information* consistently improves all metrics for every backbone, with the marking scheme (MS) contributing the majority of the gain relative to the reference solution alone (REF); combining both (REF+MS) provides a small additional improvement primarily for the strongest backbone (O3).

Instruction style under REF+MS should match the backbone. With context fixed at REF+MS, instruction choice modulates the calibration-ranking trade-off (Table 2). The strongest backbone (O3) attains the best overall accuracy and near-zero bias with the more flexible NORM prompt, whereas mid-tier models (O4-MINI) benefit from the more prescriptive STRICT prompt. This pattern suggests that stronger models can reliably and flexibly apply the marking scheme, while mid-tier models require more prescriptive guidance to reduce over-crediting and variance.

Ensembling helps. For ensembling, we aggregate five independent O3 evaluation runs under REF+MS, with results reported in Table 3. Compared to the best single run, averaging all runs reduces RMSE by ∼0.06 (1.225→1.169) and improves Kendall-τ from 0.540 to 0.578.

Model	Inst.	RMSE ↓	WTA$_{\leq 1}$ (%) ↑	Bias ≈
o3	NORM	**1.273**	**76.5**	**−0.008**
	STRICT	1.420	72.8	−0.304
	BASIC	1.348	73.0	0.165
o4-mini	NORM	1.816	67.6	0.762
	STRICT	**1.718**	**69.2**	**0.396**
	BASIC	1.817	67.6	0.724

Table 2: **Instruction ablation under REF+MS.** Three instruction styles: NORM (flexible use of the marking scheme, allowing valid alternatives), STRICT (literal adherence with penalties for deviations), and BASIC (minimal guidance).

Run	RMSE ↓	MAE ↓	kendall-τ ↑
Single (mean±std)	1.265 (±0.039)	0.981 (±0.028)	0.509 (±0.022)
Best single	1.225	0.944	0.540
MEAN	**1.169**	0.940	**0.578**
MEDIAN	1.185	**0.926**	0.540

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.

4 Downstream Utility: Best-of-N Proof Selection

Reliable evaluators should enable better selection under BoN sampling. This section answers a critical question: Does the superiority of our fine-grained evaluators, as measured by offline metrics, translate to an improvement in a downstream selection task?

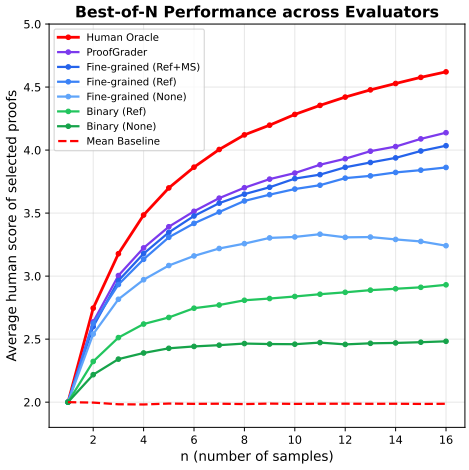

Figure 2: **Best-of-n with different *evaluators*.** Average score over 29 problems for o3 (generator) as n increases from 1 to 16.

Setup. We use o3 to generate 16 candidates per problem for 29 problems from 2025 (464 proofs total), and obtain human scores via the pipeline in §2. For each problem, 8/16 responses are graded by at least two experts with disagreements resolved through discussion. We compare our fine-grained evaluators, including PROOFGRADER (an ensemble of five o3 runs with REF+MS), to a binary baseline (Correct/Incorrect). The performance of each selection method is measured using a **best-of-n (BoN) curve**. We estimate the BoN curve using Monte Carlo subsampling (detailed can be found in Appendix §A.4). We include two key baselines: an oracle baseline and a mean baseline. The HUMAN ORACLE represents the performance ceiling, calculated as the average score of the best possible selection among the sampled n candidates. The MEAN BASELINE is calculated as the cumulative average score of the sampled n candidates. All evaluators and selectors use o3 as the backbone model.

Results. Figure 2 shows the BoN curves across multiple evaluators. Comparing the four different fine-grained evaluators from §3, we confirm that the marking scheme is a critical component and ensembling is beneficial. We further compare with binary evaluators: the o3-based models prompted to classify each proof as "Correct" or "Incorrect". In contrast to PROOFGRADER, which reaches a score of 4.14 at n=16, the binary evaluator performs only slightly better than the average score of all candidates. This limitation comes from collapsing all "correct" (or "incorrect") proofs into a single category, losing 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). We thus conclude that fine-grained scoring preserves relative ordering, making it essential for effective reward modeling.

5 Conclusion

We advance reliable evaluation of natural-language math proofs by introducing PROOFBENCH, a multi-contest, multi-year dataset with fine-grained (0–7) expert grades. Using PROOFBENCH, we map the evaluator design space and find that performance is driven by backbone model capabilities and context, with problem-specific marking schemes most impactful; simple ensembling further improves robustness. We obtain PROOFGRADER with the strongest evaluator across design choices. As a downstream test, PROOFGRADER serves as an effective reward model for best-of-n selection.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022. URL https://arxiv.org/abs/2210.10760.

[6] 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.

[7] 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.

[8] Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation, 2025. URL https://arxiv.org/abs/2506.01937.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

A Appendix

A.1 Data Information

Table 4 provides short descriptions of the contests which we sourced problems from. The data information for the collected problems are available in Table 5. Problem 4 from EGMO 2023 was intentionally removed because it contains figure in problem text.

A.2 Annotation Pipeline Details

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

Contest	Description
APMO	The Asia Pacific Mathematical Olympiad is a regional proof-based contest for high school students across Asia–Pacific. It features five challenging problems solved over a fixed time window, proctored locally.
EGMO	The European Girls' Mathematical Olympiad is an international two-day proof contest for female students, modeled after the IMO. It promotes participation and excellence among young women in mathematics.
IMO	The International Mathematical Olympiad is the premier global high school math competition. Over two days, students tackle six proof problems that test creativity, rigor, and depth.
PUTNAM	The William Lowell Putnam Mathematical Competition is a highly challenging proof exam for undergraduates in the U.S. and Canada. It consists of 12 problems emphasizing ingenuity and precise argumentation.
TST	Team Selection Tests are national-level olympiad exams used by USA to select their IMO teams. Problems mirror international difficulty and assess readiness for global competition.
USAMO	The United States of America Mathematical Olympiad is an invitational proof contest following AMC/AIME qualification. It identifies top U.S. high school problem solvers and feeds into further training and team selection.

Table 4: Brief descriptions of core contests.

Contest	2022	2023	2024	2025	Total
APMO	5	5	5	5	20
EGMO	6	5	6	6	23
IMO	6	6	6	6	24
PUTNAM	12	12	12	–	36
TST	–	6	6	6	18
USAMO	6	6	6	6	24
Total per year	**35**	**40**	**41**	**29**	**145**

Table 5: Core problem counts by contest and year.

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 145 problems with 3 solutions each, yielding 435 rubric-guided annotations. We double-score 40% of items, run periodic drift checks, and adjudicate all flagged disagreements.

Screenshots of the annotation interface are shown in Figure 3 and Figure 4

A.3 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

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

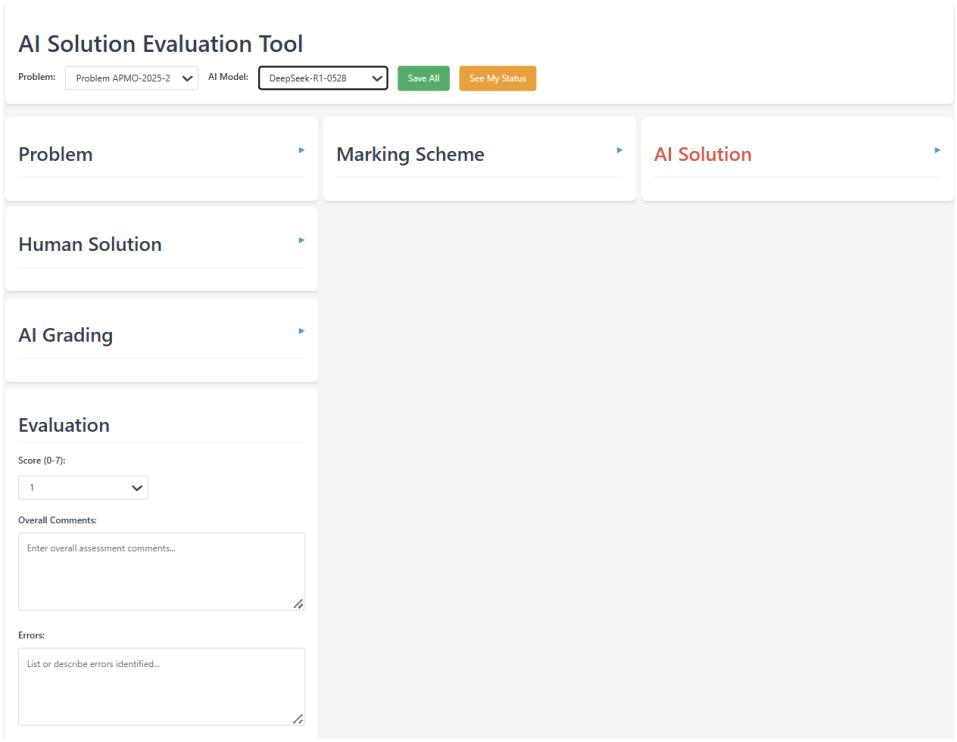

Figure 3: **View of Evaluation Platform Setup**

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 $P' = |\mathcal{P}|$:

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

$$\overline{\text{WTA}}(\leq 1) = \frac{1}{P}\sum_{p=1}^{P} \text{WTA}_p(\leq 1), \overline{\tau_b} = \frac{1}{P'}\sum_{p\in\mathcal{P}} \tau_b(p).$$

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

A.4 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 $\hat{y} = \arg\max_i \text{RM}(y_i)$, 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.

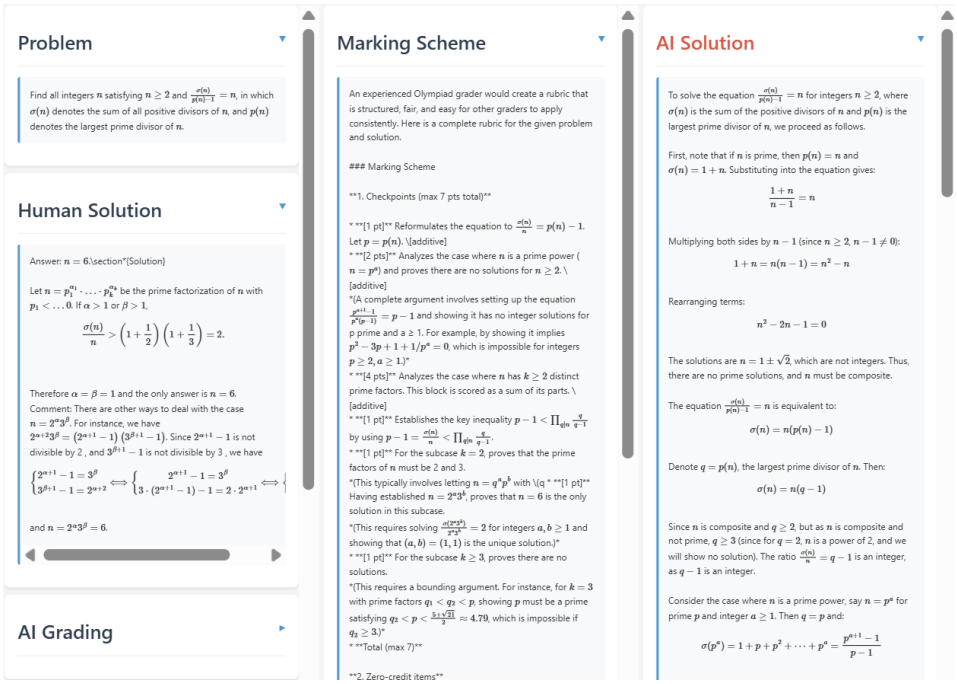

Figure 4: **View of Evaluation Platform Setup**

Best-of-n is also well-explored in other works as a metric for evaluating reward model and evaluator robustness [3, 12].

Setup. To create a testbed, we use o3 to generate 16 candidate proofs for each of 29 selected problems from 2025, resulting in 464 unique proofs. All 464 of these candidates were then scored by three human experts using the pipeline described in §2. This dataset is distinct from the **??** split and serves a different purpose. For each problem, 8 responses are graded by two experts with disagreements resolved through discussion. For this analysis, we test a selection of the fine-grained evaluators studied previously (§**??**) and several comparison-based selection strategies including Knockout tournament style selection (§**??**). The performance of each selection method is measured using a **best-of-n (BoN) curve**. Since an exhaustive evaluation over all $\binom{16}{n}$ subsets is computationally intensive, we estimate the BoN curve using Monte Carlo subsampling. For each $n \in \{1, 2, \cdots, 16\}$, we sample a large number of subsets of size n without replacement. For each subset, the evaluator selects the single proof with the highest assigned score, using the initial ordering to break ties. The performance at n is the average human score of these selected proofs, aggregated over all subsamples and all 29 problems. We also include two key baselines: an oracle baseline and a mean baseline. The HUMAN ORACLE represents the performance ceiling, calculated as the average score of the best possible selection among the first n candidates. The MEAN BASELINE is calculated as the cumulative average score of the first n candidates. All evaluators and selectors use o3 as the backbone model.

A.5 Marking Scheme Example

A.6 Generator Prompt

Default

```
Your task is to write a proof solution to the following
problem. Your proof will be graded by judges for correctness
and completeness. When you write your proof, follow these
guidelines:
```

Problem. Let $n > k \geq 1$ be integers. Let $P(x) \in \mathbb{R}[x]$ be a polynomial of degree n with no repeated roots and $P(0) \neq 0$. Suppose that for any real numbers $a_0, \ldots, a_k$ such that $a_k x^k + \cdots + a_1 x + a_0$ divides $P(x)$, the product $a_0 a_1 \ldots a_k$ is zero. *Prove* that $P(x)$ has a nonreal root.

Reference Solution. By considering any $k+1$ roots of P, WLOG assume $n = k+1$. Suppose $P(x) = (x+r_1)\ldots(x+r_n)$ has $P(0) \neq 0$. Then each polynomial $P_i(x) = P(x)/(x+r_i)$ of degree $n-1$ has ≥ 1 zero coefficient.
The leading and constant coefficients of each P_i are nonzero, leaving $n-2$ other coefficients. By pigeonhole, P_1 and P_2 share a zero coefficient position, say x^k for some $1 \leq k < n-1$.

Claim. If P_1 and P_2 both have x^k coefficient zero, then $Q(x) = (x+r_3)\ldots(x+r_n)$ has consecutive zero coefficients $b_k = b_{k-1} = 0$.

Proof. By Vieta, let $Q(x) = x^{n-2} + b_{n-3}x^{n-3} + \cdots + b_0$. The x^k coefficient of P_1, P_2 being zero means $r_1 b_k + b_{k-1} = r_2 b_k + b_{k-1} = 0$,
hence $b_k = b_{k-1} = 0$ (using r_i nonzero, distinct). □

Lemma. If $F(x) \in \mathbb{R}[x]$ has two consecutive zero coefficients, it cannot have all distinct real roots.

Proof 1 (Rolle). Say x^t, x^{t+1} coefficients are zero. $\cdots$ But $F^{(t)}(x)$ has a double root at 0, contradiction. □

Proof 2 (Descartes). Real roots are bounded by sign changes in $F(x)$ plus sign changes in $F(-x)$. $\cdots$
With b nonzero coefficients and z runs of zeros, real roots $\leq b - 1 + z \leq \deg F$. Two consecutive zeros make this strict. □

Marking Scheme (max 7 pts). *Checkpoints (additive):*
(1) [1pt] Problem reduction to $n = k+1$ and setup. Alt: complete proof for $k = 1$.
(2) [2pts] Pigeonhole: n polynomials, $n-2$ internal coefficient positions; two share a zero position.
(3) [2pts] Deduce consecutive zeros: from $[x^m]P_1 = [x^m]P_2 = 0$ and $P_i = (x+r_i)Q$, show $b_m = b_{m-1} = 0$.
(4) [2pts] Prove lemma (2pts for complete Rolle or Descartes proof; 1pt for partial).
Deductions: Cap 6/7 if no reduction justification; cap 5/7 if lemma flawed; cap 3/7 if stops after PHP; -1pt for minor gaps (e.g., not using $r_1 \neq r_2$).
Zero credit: Unjustified WLOG; merely stating theorems; specific examples only; noting $P(0) \neq 0 \Rightarrow$ nonzero roots.

Figure 5: Example problem (USAMO 2025 P2) with its reference solution and marking scheme.

```
- You are creating a proof, not a proof outline. Each step
should be carefully explained and documented. If not properly
explained, the judge will assume that you cannot explain it,
and therefore decrease your grade.
- You can use general theorems and lemmas, but only if they
are well-known. As a rule of thumb: if the result has a name
and is famous enough to have a Wikipedia page or something
similar to describe it, it is allowed. Any result from papers
that would not be taught in high school or low-level bachelor
courses in mathematics should not be used. Any use of such
results will immediately give you a zero grade.
- Do not skip computation steps in your proof. Clearly
explain what transformations were done and why they are
allowed in each step of a calculation.
- Your proof should be self-contained.
- If you are not sure about a specific step, or do not
know how to prove an intermediate result, clearly state
this. It is much preferable to indicate your uncertainty
rather than making incorrect statements or claims.

FORMATTING GUIDELINES:
- You should write Markdown with LaTeX math. Do NOT use
code fences (no ```).
- You should use correct LaTeX notation to write equations
and mathematical symbols. You should encompass these
equations in correct delimiters ("\\(" and "\\)" for
inline math, "\\[" and "\\]" for block math) to enhance
the clarity of your proof. **Do not use any unicode
characters.**
```

```
- For multi-line derivations, wrap an aligned block
INSIDE display math.
- Do not use other LaTeX environments or packages.

PROBLEM: {problem}
```

A.7 Marking Scheme Generation Prompt

Marking Scheme

```
You are an experienced Olympiad grader.
*Treat the official solution as fully correct and
authoritative; do not claim it contains errors or gaps.*
Your task is to write a complete, grader-friendly rubric for
the problem and official solution below. The rubric must map
a student's proof to an integer score from **0 to 7**.

---

### INSTRUCTIONS

Produce the marking scheme in **exactly** the following three
sections.
1. **Checkpoints (max 7 pts total)**

  * Break the solution into logically independent checkpoints
  with **integer** point values.

    * Allocate **>= 4 pts** to the main idea/critical steps;
    **<= 3 pts** to routine work.
  * If two items are mutually exclusive (solve the *same*
  logical gap), **nest** them and write **\award the larger
  only"**.
  * **Parallel solution paths (non-additive):**
    * If the official solution (or a student submission)
    admits more than one legitimate approach, write
    **parallel checkpoint chains** labeled \Chain A
    / Chain B / ... (idea: ...)".
    * **Start this section with a bold rule:** **Score
    exactly one chain | take the **maximum** subtotal
    among chains; do **not** add points across chains.**
    * Within a chain, checkpoints may be \[additive]. For
    steps **shared across chains** (e.g., a lemma usable by
    multiple approaches), place them under a \Shared
    prerequisites" group with **\[max k]**, and state
    **\count at most once regardless of chain."**
  * For every bullet (or group), append either
  **\[additive]** or **\[max k]** to make the scoring
  rule unambiguous.
  * Finish with a one-line **\Total (max 7)"** check that is
  consistent with the non-additivity rule.
  * Never demand cosmetic labels or specific variable names
  unless essential to the logic.

2. **Zero-credit items**

  * List common arguments or observations that **earn 0
```

```
  points** (e.g., conjectures without proof|especially in
  geometry, routine restatements of the problem, or
  dead-ends).

3. **Deductions**

  * Bullet each typical mistake with a **flat** penalty
  (**{1**, **{2**, or **\cap at x/7"**).
  * Apply **at most the single largest** deduction; never
  reduce a score below 0.
  * Target logic gaps, invalid claims, circular reasoning,
  or contradictions. Cosmetic slips (notation, arithmetic,
  wording) do **not** trigger deductions unless they break
  validity.
  * If a student gives **multiple distinct proofs**, **grade
  the best one only** (no stacking). If proofs contain
  **contradictory claims**, apply an appropriate deduction
  (e.g., **cap at 5/7**).

### IMPORTANT REQUIREMENTS

* Use **concise bullets** | no prose exposition of the
official solution.
* **Arithmetic sanity checks:** per-chain checkpoint sums
and \[max k]
caps must make it impossible to exceed **7** overall; at
least one chain must allow a perfect **7/7**.
* Do **not** introduce, \fix," or critique the official
solution.
* Avoid over-fragmenting: do not split routine algebra
into 3+ separate 1-pt bullets.
* Keep notation consistent with the official solution;
define any new symbols you introduce.

---

### PROBLEM

{problem}

### OFFICIAL SOLUTION

{solution}
```

A.8 Evaluator 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 > problem
constraints > marking scheme alignment > 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}
```