Scaling Generative Verifiers For Natural Language Mathematical Proof Verification And Selection

Sadegh Mahdavi[1,2*], **Branislav Kisacanin**[1,3], **Shubham Toshniwal**[1], **Wei Du**[1], **Ivan Moshkov**[1]
George Armstrong[1], **Renjie Liao**[2†], **Christos Thrampoulidis**[2†], **Igor Gitman**[1†]

[1]NVIDIA, [2]University of British Columbia, [3]Institute for AI R&D of Serbia

Abstract

Large language models have achieved remarkable success on final-answer mathematical problems, largely due to the ease of applying reinforcement learning with verifiable rewards. However, the reasoning underlying these solutions is often flawed. Advancing to rigorous proof-based mathematics requires reliable proof verification capabilities. We begin by analyzing multiple evaluation setups and show that focusing on a single benchmark can lead to brittle or misleading conclusions. To address this, we evaluate both proof-based and final-answer reasoning to obtain a more reliable measure of model performance. We then scale two major generative verification methods (GenSelect and LLM-as-a-Judge) to millions of tokens and identify their combination as the most effective framework for solution verification and selection. We further show that judgement prompt choice significantly affects judging performance, but reinforcement learning can reduce this sensitivity. However, despite improving proof-level metrics, reinforcement learning does not enhance final-answer precision, indicating that current models often reward stylistic or procedural correctness rather than mathematical validity. Our results establish practical guidelines for designing and evaluating scalable proof-verification and selection systems.

1 Introduction

Large Language Models (LLMs) have achieved remarkable progress in mathematical reasoning tasks, reaching near-saturation on high-school competition benchmarks such as MATH [6] and AIME [15]. This success is driven in large part by the simplicity of evaluating final-answer problems: correctness can be checked automatically, either through string matching or with lightweight equivalence checks [16]. This makes such tasks well-suited for reinforcement learning with verifiable rewards (RLVR) [17] [17], enabling steady improvements in model performance.

Closer inspection, however, reveals a major gap: LLMs often arrive at correct answers through flawed reasoning [5]. For advancing towards harder competitions such as the IMO or USAMO, where solutions require rigorous proofs rather than just final answers, reliable proof verification becomes essential. Unlike final answers, verifying natural-language proofs is inherently more complex, requiring deeper understanding of logical structure and mathematical argumentation. Developing reliable verification methods in this context could enable RLVR-style approaches to improve proof generation, analogous to their success with final-answer tasks. This could involve building absolute verification systems that determine whether a proof is correct, providing precise reward signals for policy-gradient RL approaches such as GRPO [17], or designing selection and verification mechanisms that leverage test-time computation to improve over standard pass@1 methods, with outputs usable for self-improvement or expert iteration [24, 4].

Due to the success of LLMs in acting as verifiers [25, 9], we focus on LLM-based generative verification methods. Particularly, our contributions are as follows: We start by establishing a robust

[*]Work done during internship at NVIDIA. [†]Corresponding Authors.

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

evaluation setup which shows that existing datasets have significant limitations. In addition to that, we explore various prompting options including instruction variation as well as adding rubrics. We then explore different ways to scale test-time compute, namely ensembling, scaling the number of parallel judgements [18, 27] and performing GenSelect [20]. Combining all this together we arrive at a state-of-the-art proof judge method and confirm its effectiveness by applying it on both final answer problems to select the best solution as well as on proof problems to select the best proof.

2 Related Work

Test-Time Scaling for Mathematical Reasoning. It has been shown that test-time scaling improves performance on mathematical reasoning tasks. From early results on sampling multiple trajectories from LLMs and majority-voting-based methods [21] to more recent breakthroughs such as long reasoning [8], which scale the length of the chain-of-thought in the model. More recent approaches involve generating multiple solutions and selecting the best one using a judge model via a generative verifier [11, 20, 3], or iteratively refining solutions using feedback from a judge model [13]. Here, we build upon GenSelect [20], and LLM-as-a-Judge ideas and combine them to achieve even higher performance on mathematical reasoning tasks, particularly the proof-based ones.

Mathematical Reasoning Beyond Final Answers. Recent research has focused on advancing mathematical reasoning beyond simply producing final answers. Datasets such as PRM800K [10] and ProcessBench [26] pioneered the training and evaluation of large language models (LLMs) on process-level errors, moving past traditional final-answer-based assessment. More recently, efforts have targeted proof-based and challenging mathematical problems, including those from the International Mathematical Olympiad (IMO). Mahdavi et al. [14] evaluated state-of-the-art LLMs on proof-based tasks, revealing common pitfalls such as overgeneralization from limited examples, circular reasoning, and fabricating non-existent theorems. Dekoninck et al. [3] further contributed to this area by introducing new benchmarks for the rigorous evaluation of LLMs for the verification and selection of proofs, concluding that LLMs have approached human-level performance in the evaluation of proofs.

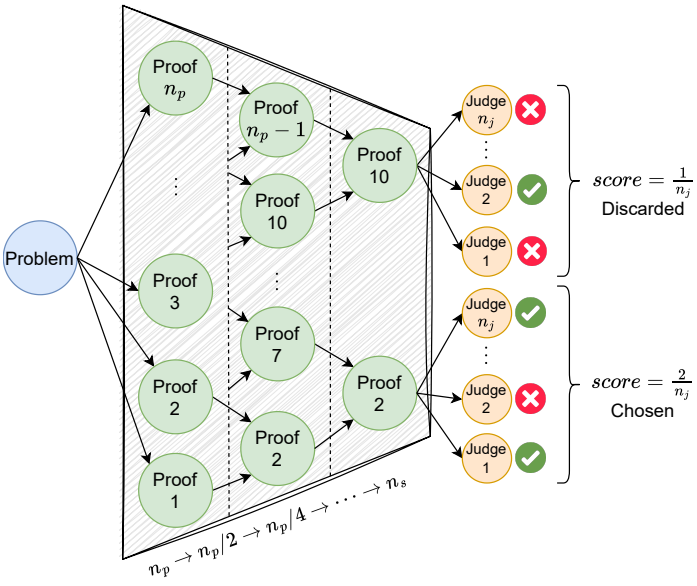
Reward Models for Hard-to-Verify Tasks. In contrast to tasks with easily verifiable answers, proof verification in natural language is an inherently challenging task, even for humans. Recent works have explored training reward models to tackle such hard-to-verify tasks. While there is no publicly available work on training reward models for proof verification, there have been various works in other domains such as open-ended writing. The current mainstream approach is to train generative verifiers or use off-the-shelf LLMs to judge the correctness or ranking of model-generated solutions, and use the verifiers to guide the generation of better solutions. Team et al. [19] use a self-critique-based approach to generate responses and act as its own critique to judge the response given a predefined rubric for the task. Lu [12] also use a self-critique approach to rank the generated model responses of the underlying LLM for open-ended writing tasks. These rankings are then used to produce rewards for training a reward model, which is then used to further finetune the LLM using reinforcement learning.

3 Method

We first explain the datasets we use, and our test-time scaling approach, then, we justify different prompts, alternative test-time scaling methods, and variable choices via ablation studies.

3.1 Scaling test-time compute

Choosing the evaluation set carefully is crucial. We describe the datasets we use for training and evaluation. We find that the choice of evaluation set is crucial for developing and benchmarking proof judgment models. We identify two sources of challenge in building a reliable evaluation set. First, human label noise can be significant, making it difficult to obtain large-scale reliable ground truth judgment labels. Second, model and problem imbalances in the dataset may lead models to exploit spurious correlations to achieve high accuracy without genuinely understanding the mathematical content of the proofs. For instance, we show that a text embedding-based multilayer perceptron (MLP) trained on the training set of the Open Proof Corpus (OPC) dataset outperforms Claude-Sonnet-4 by more than 5% (see Appendix A for more details). In this paper, we mainly experiment with three datasets:



Challenge-19 Results

n_p	n_s	n_j	Acc (%)
64	1	–	88.81
128	1	–	91.44
256	1	–	92.10
128	16	32	91.44
256	16	32	**96.05**
512	16	32	94.07
maj@512			82.23
pass@1			47.07

AIME 2025 Results

n_p	n_s	n_j	Acc (%)
128	1	–	97.5
128	16	32	**100**
maj@128			97.08
pass@1			91.26

Figure 1: *Left:* Overview of our proposed test-time scaling method for proof selection. Given a question, we first generate n_p candidate proofs by sampling from a language model. Then we run a knockout GenSelect tournament to select the best proof among the candidates using the same language model to select the top n_s proofs. We estimate the correctness score of each proof using LLM-as-A-Judge by sampling n_j judgements for each proof. Finally, we select the proof with the highest average correctness score as the final proof to solve the problem. *Right:* The tables show accuracy results for different configurations of our method on GPT-OSS-120B, averaged across 8 seeds on AIME-2025 and Challenge-19 (a set of challenging final-answer problems from HMMT, AIME, and CMIMC) combining LLM-as-a-Judge with GenSelect performs substantially better than GenSelect alone, or majority voting.

- A collection of proofs from USAMO 2025, IMO 2025, and IMC 2025, with human-labeled judgements from MathArena [2], as well as Gemini and OpenAI's solutions to the IMO 2025. The majority of the proofs are graded by at least two human graders, or are officially accepted solutions. Although this dataset has the highest label reliability, it may still suffer from the imbalanced issue mentioned earlier
- We select 19 most challenging problems from AIME 2024, HMMT 2024, and CMIMC 2025 with integer final answers, and use them to construct a balanced evaluation set (calling the dataset Challenge-19). We only keep the problems for which GPT-OSS-120B [1] with high-reasoning mode generates at least one correct final answer and has a solve rate lower than 70%. Then, we generate an equal number of correct-final-answer and incorrect-final-answer solutions for each problem using GPT-OSS-120B and Qwen3-235B-A22-Thinking-2507. The solutions in this dataset are particularly useful since we can monitor the precision of a judge on this dataset. That is, if a judge predicts a proof to be correct, precision computes how likely is it that the proof has the correct final answer. Monitoring precision allows us to distinguish between judges that rely primarily on superficial features (e.g., stylistic cues) versus those that genuinely assess the mathematical content. Because of its balanced construction, this final-answer dataset provides a robust benchmark for precision-based evaluation.
- The pass@n subset from the Open-Proof-Corpus dataset consisting of 60 problems with 8 proofs per problem generated by the OpenAI o4-mini model. The task is to select the best proof among 8 generated proofs.

Test-Time Scaling via GenSelect Tournaments and LLM-as-a-Judge. Given the success of generative verifiers via pairwise comparison [20, 3], and the sucess of LLM-as-a-Judge [22, 25, 18], we propose to combine both methods to further scale the performance of proof selection at test time.

First, we generalize the GenSelect and LLM-as-a-Judge frameworks into a unified two-stage process. Given a problem, we first generate n_p candidate proofs by temperature sampling from a language model. Then we run a knockout GenSelect tournament to select top n_s proofs among the candidates using the same language model. Finally, we run LLM-as-a-Judge to estimate the correctness score of each proof by sampling n_j judgements for each proof, and select the proof with the highest average correctness score as the final proof to solve the problem.

Figure 1 illustrates our proposed method and accuracy results for different configurations on GPT-OSS-120B. We find that this combined method significantly outperforms majority voting, and consistently matches or outperforms GenSelect [20]. Notably, we achieve 100% accuracy on AIME-2025 across all 8 runs with different random seeds using GPT-OSS-120B.

3.2 Ablation studies

LLM-as-a-Judge Prompt Ablation. We evaluate three prompts for LLM-as-a-Judge: a general-purpose prompt adapted from ProcessBench [26], and two proof-specific prompts from OpenProof-Corpus [3] and a Gemini Agent [7]. To standardize their outputs, we only modify the formatting to use XML tags, leaving the prompt content unchanged. We refer to these prompts as `General Summary`, `OPC`, and `GIMO`. Figure 2 (right) reports majority@5 results on both proof-level and final-answer judgements. The results show that prompt choice has a substantial impact on LLM-as-a-Judge performance: `General Summary` achieves high recall but low precision, `GIMO` achieves high precision but low recall, and `OPC` strikes a balance between the two. Based on this trade-off, we adopt `OPC` as the default prompt in our main experiments.

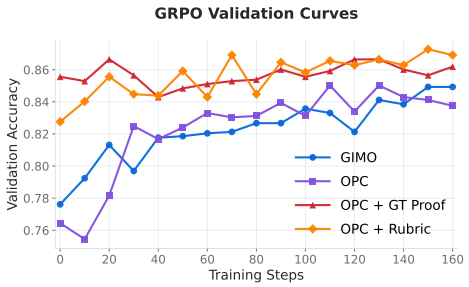
RL eliminates prompt variation. Given the high variation in the performance of different prompts, we experiment with using RL to fine-tune a language model to be a better judge. We use the OPC training set to train a Qwen3-30B-A3B-Thinking-2507 model using Group Policy Optimization (GRPO) [17] with a binary reward, measuring whether the model judgment matches the ground truth judgement. Figure 2 shows the training curve of the RL training with different prompts. We observe that RL training significantly improves the performance on all the prompts and eliminates the variation in performance across `OPC` and `GIMO` prompts. Although the `OPC` prompt initially exhibits lower performance compared to the `GIMO` prompt, RL training leads to convergence to similar results.

RL does not improve final-answer precision, despite improving all the proof metrics. While we observe that RL improves performance on the proof judgement (Figure 2 Right), it does not improve the precision on the final-answer problems. We hypothesize that the proof performance gain is likely due to the model being able to identify incorrect style of proofs such as missing justifications rather than truly reasoning about the mathematical content of the proofs. This will likely require scaling up the training data with challenging problems, as well as scaling up the reinforcement learning training to larger number of steps to gain more significant gains on the mathematical content judgement.

Adding rubric does not significantly improve performance. We evaluate whether including a ground-truth rubric in the prompt during verification improves the model's ability to judge proofs and enables more effective auto-labeling. Overall, adding the rubric does not lead to a substantial performance gain in proof judgment, except for improvement in final-answer precision (see Figure 2 Right). Surprisingly, models trained without a rubric outperform those trained with one on proof-based judgment, even though they underperform on the OPC validation curves shown in Figure 2. Our best-performing model is, in fact, trained without a rubric but evaluated with a rubric at test time.

Ensembling different judges does not outperform the best single judge. As an alternative scaling strategy, we evaluate ensembles formed by averaging the correctness scores of different judges (i.e., combining half the scores from one judge and half from another). We test this approach on the OPC-pass@n proof selection task and find that ensembling generally fails to surpass the best individual judge. The only exception occurs when one judge is a model trained on the OPC dataset with reinforcement learning (see Figure 3 in the Appendix), which we attribute to adaptation to in-distribution training data.

See Appendix C for more ablation studies on the number of sampled judgements,and the impact of different base models.



Prompt	Proofs			Final Answers*	
	Prec	Rec	F1	Prec	Rec
General Summary	18.28	99.89	30.91	58.21	74.86
GIMO	57.58	87.06	69.28	64.85	15.53
OPC	22.88	99.96	37.23	60.41	67.43
OPC (Rubric)	24.46	99.89	39.30	92.25	77.45
OPC RL	54.03	94.57	68.75	62.59	17.18
GIMO RL	49.66	95.81	65.40	66.29	22.65
OPC RL (Rubric)	39.72	98.98	56.69	98.15	50.02
OPC RL (GT-Proof→Rubric)	39.09	98.64	55.99	98.12	49.06
OPC RL (No-Rubric→Rubric)	59.74	95.09	73.35	99.31	21.27

Figure 2: *Left:* Training curves for RL training of Qwen3-30B-A3B-Thinking-2507 on the OPC dataset. (1) Different prompts converge to similar performance after RL training. (2) Including the ground-truth proof or rubric in the prompt improves validation performance by 2–3%. (3) Training with both rubric and ground-truth proof achieves comparable performance. *Right:* Majority@5 results for the Math Proof Judge evaluation using Qwen3-30B-A3. See Table 4 in the Appendix for full results, additional models, and prompt ablations. *For final-answer problems, precision and recall are computed based solely on the correctness of the final answer.

4 Conclusion

In this work, we addressed the critical challenge of verifying and selecting natural-language mathematical proofs using LLMs. We showed a unified test-time scaling approach that combines GenSelect tournaments with LLM-as-a-Judge evaluation performs the best at scale. We made a surprising finding: while reinforcement learning eliminates prompt variation and improves proof-level metrics, it does not improve final-answer precision, suggesting current approaches may rely more on recognizing stylistic features rather than deep mathematical understanding.

Future Directions. An important direction for future work is to focus on problems at the frontier of current LLMs' solving capabilities. By carefully selecting such challenging problems and rigorously labeling them, we aim to create high-quality benchmarks that drive further improvements in proof-judgment and problem-solving models. We also plan to leverage RL to further scale up the performance of LLM-based judges and solvers, as well as explore other scaling methods such as process supervision, where intermediate reasoning steps are supervised rather than the final outcome.

References

[1] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

[2] Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, February 2025. URL `https://matharena.ai/`.

[3] 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.

[4] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

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

[6] 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. *CoRR*, abs/2103.03874, 2021. URL `https://arxiv.org/abs/2103.03874`.

[7] Yichen Huang and Lin F Yang. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*, 2025.

[8] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[9] Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.

[10] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

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

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

[13] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

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

[15] OpenAI. Gpt-5 system card. Technical report, OpenAI, 2025. URL `https://cdn.openai.com/gpt-5-system-card.pdf`. Accessed: 2025-09-19.

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

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

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

[19] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

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

[21] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

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

[23] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chen Zhang, Chuanqi Tan, Daxin Jiang, Dongdong Zhang, Fandong Meng, Fuli Luo, Guolin Ke, Hang Yan, Hao Tian, Hong Chen, Hongbin Zhang, Hongyi Yuan, Huanbo Luan, Hui Liu, Jiayi Wang, Jidong Zhai, Jindong Chen, Jinhui Liu, Jinqiu Sun, Jiong Cai, Jizhou Huang, Jun Xie, Junyang Lin, Kai Chen, Lei Li, Lei Wang, Liang Wang, Lianxin Jiang, Linjun Shou, Liyang Lu, Lu Chen, Lu Wang, Mengdi Wang, Ming Gong,

Ming Yan, Ming Zhou, Nan Duan, Pengcheng Shi, Qian Liu, Qiang Qu, Qianqian Dong, Qiang Wang, Qingsong Wen, Qun Liu, Renjie Zheng, Ruofei Zhang, Rui Wang, Shuming Ma, Shuo Ren, Shuxin Zheng, Songlin Hu, Tao Yu, Tianyu Liu, Tong Xiao, Wei Li, Wei Wang, Wenbin Jiang, Wenhao Huang, Wenxuan Wang, Xiaodong Liu, Xiaofei Sun, Xiaohui Yan, Xiaojun Wan, Xiaolei Wang, Xiaoyan Zhu, Xin Jiang, Xing Xie, Xinyan Xiao, Xipeng Qiu, Xu Tan, Xuefeng Bai, Xuejun Ma, Xun Wang, Yan Gao, Yang Liu, Yanghua Xiao, Yanyan Lan, Yao Meng, Yaqing Wang, Yichong Xu, Yifan Wang, Yihong Chen, Yiming Cui, Yining Wang, Yiqun Liu, Yixuan Su, Yongbin Li, Yongfeng Zhang, Yongxin Lian, Yu Bai, Yu Sun, Yuan Yao, Yuanhang Zhang, Yucheng Wang, Yufan Jiang, Yujie Zhang, Yujing Wang, Yunchang Yang, Yunzhi Yao, Zhenzhong Lan, Zhiwei Zhao, Zhiwu Lu, Zhiye Liu, Zhiwei Wang, Zhiyuan Liu, Zihang Jiang, and Zixuan Zhang. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL `https://arxiv.org/abs/2505.09388`.

[24] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

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

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

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

A Pitfalls of imbalanced evaluation set

We demonstrate that when an evaluation set is imbalanced (due to model choice or problem hardness imbalance), even simple heuristics can cause large variations in measured performance. As a case study, we consider the training and test sets of the Open-Proof-Corpus (OPC) [3]. Using the training set, we train a simple two-layer Multi-Layer Perceptron (MLP) binary classifier on text embeddings of (problem, proof) pairs. The embedding model is a small Qwen3-0.6B [23] embedding model. Using this setup, we achieve 75.34% accuracy on the OPC test set.

To further illustrate this phenomenon, we implement a simple heuristic: classify a proof as incorrect if it contains the word "triangle," correct if it contains the phrase "`---`," and incorrect otherwise. Surprisingly, this heuristic achieves 65.07% accuracy on the test set, outperforming Qwen3-8B and GPT-4.1. Its success stems from the fact that LLMs often generate incorrect geometry proofs, so predicting all geometry proofs as incorrect already yields high accuracy. Moreover, OpenAI thinking models tend to produce more correct proofs than other LLMs, and they use "`---`" to separate different proof sections. Consequently, despite having no real understanding of proof correctness, this heuristic attains high accuracy due to test-set imbalance.

A similar issue has been observed by Shi and Jin [18] in the context of training judge models for final-answer judgement. They show that providing RL training sets composed entirely of correct or incorrect problems leads to lower performance than using sets where each model produces both correct and incorrect answers. While their observation pertains to training, we find that analogous effects occur on validation sets of the judge as well.

Finally, we observe that GPT-OSS-120 achieves 87.67% accuracy on the OPC test set (maj8), whereas the estimated human error rate on this set is 9.6% [3]. This discrepancy indicates that the OPC test set is not suitable for evaluating strong proof graders. Consequently, in the main experiments, we only use the training set to train a weaker model, Qwen3-30B-A3, as a proof grader.

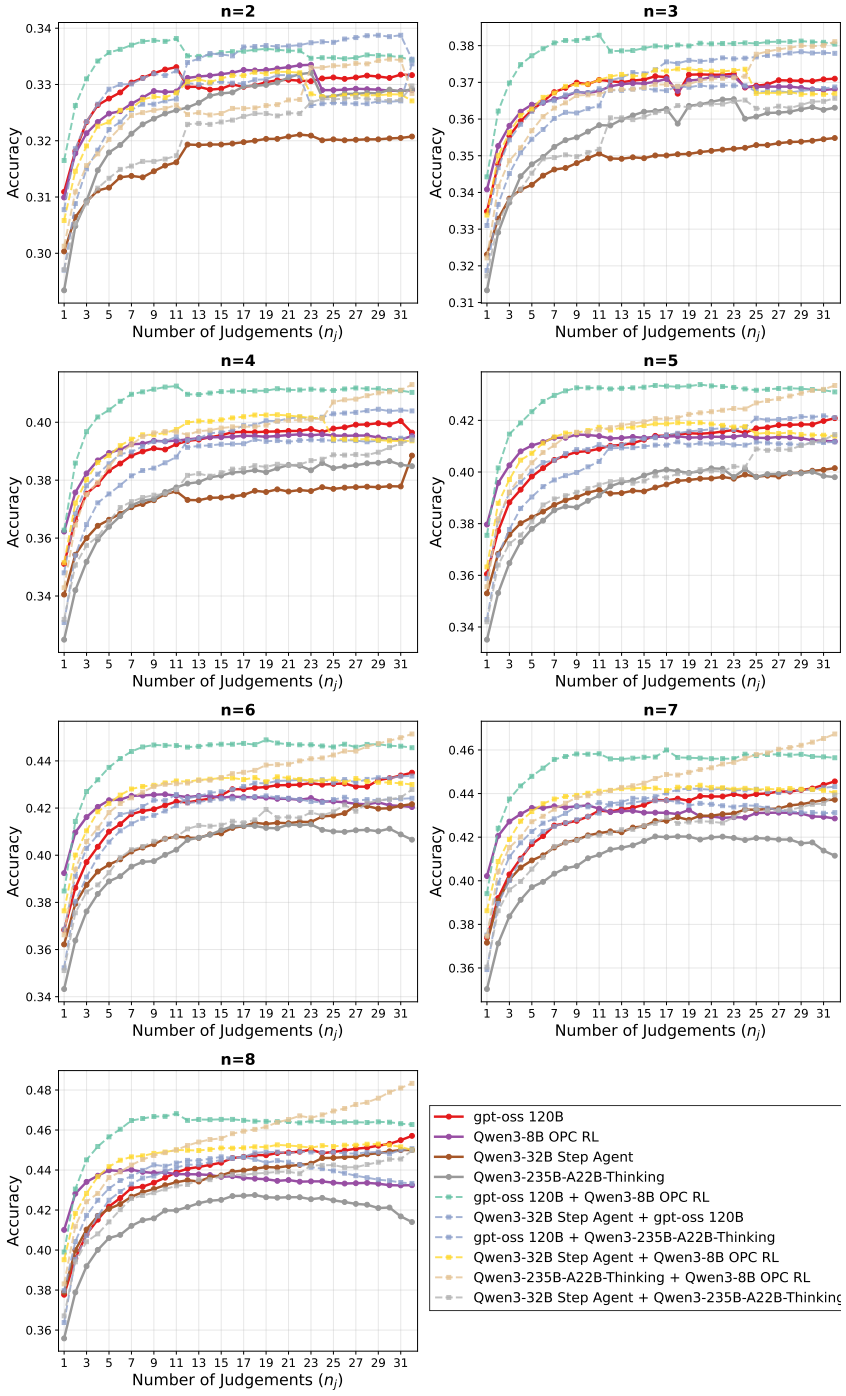


Figure 3: LLM-as-a-Judge performance on selecting the best proof among 8 generated proofs from OPC pass@n subset. Higher number of judgements leads to better performance, plateauing around 20-30 judgements. Furthermore, ensembling multiple models does not lead to any performance gain, and often the result does not exceed the strongest LLM, except for ensembling an 8B model trained with RL on the training set of OPC. We suspect that the improvement comes from the fact that the 8B model trained with RL is trained on the same distribution as the test set, and therefore is complementary to other LLMs.

Table 1: Benchmarking LLMs as proof graders. All LLM numbers are taken from Dekoninck et al. [3]. We observe that a simple MLP classifier and a simple heuristic can outperform many LLMs.

Judge	pass@1
Human	90.4
Gemini-2.5-Pro	85.4
OPC-R1-8B	83.8
OpenAI-O4-Mini	83.8
OpenAI-O3	83.1
Gemini-2.5-Flash	82.7
Qwen3-235B-A22	81.8
DeepSeek-R1	80.9
MLP Classifier	75.3
Qwen3-30B-A3	74.0
DeepSeek-R1-Qwen3-8B	70.7
Claude-4-Sonnet	70.6
Format Heuristic	65.07
Qwen3-8B	64.4
GPT-4.1	61.4
Baseline	53.2

Table 2: List of challenging final-answer problems from HMMT, AIME, and CMIMC used in the Challenge-19 dataset. All problems have an integer final answer.

Index	**Problem ID**
1	CMIMC 2025 P5
2	CMIMC 2025 P7
3	CMIMC 2025 P18
4	CMIMC 2025 P34
5	AIME 2024 P7
6	AIME 2024 P14
7	AIME 2024 P26
8	HMMT Nov 2024 HMIC 1
9	HMMT Nov 2024 Guts 19
10	HMMT Nov 2024 Theme 7
11	HMMT Nov 2024 Theme 9
12	HMMT Feb 2024 Guts 28
13	HMMT Feb 2024 Guts 29
14	HMMT Feb 2024 Combinatorics 1
15	HMMT Feb 2024 Combinatorics 4
16	HMMT Feb 2024 Combinatorics 9
17	HMMT Feb 2024 Combinatorics 10
18	HMMT Feb 2024 Algebra 10
19	HMMT Feb 2025 Team 4

B Experimental Details

For all the models in this work, we allocate a $100K$ completion length, and use the recommended sampling parameters by the model providers. For GPT-oss models, we sample with temperature 1.0 and top-p 1.0, for Qwen models, we sample with temperature 0.6, top-p 0.95 and top-k 20. For other models, we use temperature of 0.7 and topp 0.95.

C Further Ablation Experiments

GenSelecting judgements improves absolute judgement performance. We further experiment with scaling the inference time compute through GenSelect on the list of judgements to improve

absolute judgement performance. That is, given a problem, a proof, and n_j judgements, we use GenSelect to select the best judgement among the n_j judgements and use that judgement to evaluate the proof. We run this genselect for n_j seeds and report the majority@5 results in Table 3. We observe improvements in precision and F1 for both GPT-OSS-120 and Qwen3-30B-A3-Thinking, while recall drops slightly. However, when incorporating GenSelect into proof-selection (i.e., selecting the best proof among candidates), we observe no performance improvement. This suggests that GenSelect is particularly beneficial for absolute judgement selection, where choosing a threshold is not possible, but does not provide gains in the proof-selection setting.

Table 3: GPT-OSS-120 and Qwen3-30B-A3-Thinking evaluation. We report precision, recall, and F1 for proof problems, and precision for final-answer problems.

Model	**Judgement**	**Proof Problems**			**Final-Answer Problems**
		Precision	Recall	F1	Precision
GPT-OSS-120	LLM-as-Judge	46.28	**99.17**	63.10	86.18
GPT-OSS-120	Judge GenSelect	**54.46**	91.21	**68.19**	**90.13**
Qwen3-30B-A3-Thinking	LLM-as-Judge	22.08	**99.96**	37.23	60.41
Qwen3-30B-A3-Thinking	Judge GenSelect	**30.65**	94.49	**46.29**	**66.26**

Table 4: Majority@5 Results for Math Proof Judge Evaluation, averaged across 32 seeds.

Prompt	Proofs				Final Answers		
	Precision	Recall	F1	Avg Tokens	Precision	Recall	Avg Tokens
Qwen3-30B-A3B-Thinking-2507							
General Summary	18.28 ± 0.28	99.89 ± 0.45	30.91 ± 0.41	8587	58.21 ± 1.37	74.86 ± 2.19	11813
GIMO	57.58 ± 2.30	87.06 ± 2.95	69.28 ± 2.09	8503	64.85 ± 5.88	15.53 ± 2.39	10025
Gemini 1	56.01 ± 1.95	88.72 ± 2.34	68.65 ± 1.87	8664	62.04 ± 4.59	17.14 ± 1.83	10352
Gemini 2	54.69 ± 2.53	88.98 ± 2.31	67.70 ± 2.17	8779	65.31 ± 3.97	18.55 ± 1.74	10716
OPC	22.88 ± 0.57	99.96 ± 0.26	37.23 ± 0.76	6327	60.41 ± 1.52	67.43 ± 2.30	8609
OPC (Rubric)	24.46 ± 0.51	99.89 ± 0.45	39.30 ± 0.67	6046	92.25 ± 1.44	77.45 ± 2.25	10211
Prompt 1 (Ablation)	19.43 ± 0.30	100.00	32.54 ± 0.42	7276	58.23 ± 1.40	78.39 ± 1.70	9793
Prompt 2 (Ablation)	20.68 ± 0.35	100.00	34.27 ± 0.49	6856	59.21 ± 1.21	73.08 ± 1.91	9323
Prompt 3 (Ablation)	21.99 ± 0.47	99.36 ± 0.97	36.01 ± 0.65	7774	61.42 ± 1.32	76.27 ± 1.92	10994
Prompt 4 (Ablation)	23.77 ± 0.46	99.96 ± 0.26	38.41 ± 0.60	6634	60.32 ± 1.90	64.41 ± 2.73	8995
Prompt 5 (Ablation)	16.65 ± 0.21	99.96 ± 0.26	28.54 ± 0.32	7757	56.64 ± 1.17	81.63 ± 1.98	11123
Prompt 5 (Rubric)	22.27 ± 0.38	96.53 ± 1.15	36.18 ± 0.55	5999	98.54 ± 0.64	94.76 ± 1.35	11273
Prompt 6 (Rubric)	27.28 ± 0.69	99.02 ± 1.15	42.77 ± 0.88	5656	95.21 ± 1.68	74.61 ± 2.77	10182
OPC RL	54.03 ± 1.81	94.57 ± 2.44	68.75 ± 1.76	5271	62.59 ± 3.59	17.18 ± 1.50	6070
OPC RL (Rubric)	39.72 ± 1.06	98.98 ± 1.32	56.69 ± 1.15	5380	98.15 ± 0.68	50.02 ± 2.60	8238
OPC RL (Train GT Proof, eval Rubric)	39.09 ± 1.24	98.64 ± 1.25	55.99 ± 1.38	5362	98.12 ± 1.28	49.06 ± 2.67	8216
OPC RL (Train w/o rubric, eval rubric)	59.74 ± 2.53	95.09 ± 2.33	73.35 ± 2.27	4997	99.31 ± 1.59	21.27 ± 1.95	6897
GIMO RL	49.66 ± 1.50	95.81 ± 2.21	65.40 ± 1.52	6008	66.29 ± 3.47	22.65 ± 1.87	7041
GPT-OSS-120B							
General Summary	41.82 ± 1.44	95.47 ± 1.73	58.15 ± 1.57	17443	86.32 ± 2.53	52.67 ± 2.31	30709
GIMO	94.11 ± 3.26	45.09 ± 3.07	60.90 ± 2.96	15599	94.60 ± 12.18	2.00 ± 1.41	24512
Gemini 1	96.22 ± 2.78	54.26 ± 3.95	69.29 ± 3.24	12933	83.50 ± 26.90	1.84 ± 1.09	21280
Gemini 2	94.82 ± 3.08	55.13 ± 4.36	69.62 ± 3.62	12977	85.78 ± 19.59	2.86 ± 1.33	22172
OPC	46.28 ± 1.24	99.17 ± 1.08	63.10 ± 1.22	16807	86.18 ± 2.53	50.29 ± 1.89	32529
OPC (Rubric)	44.32 ± 1.29	99.89 ± 0.45	61.39 ± 1.25	10502	99.61 ± 0.79	49.22 ± 2.70	16848
Prompt 1 (Ablation)	47.65 ± 1.40	97.96 ± 1.12	64.10 ± 1.38	11267	82.23 ± 2.38	37.84 ± 2.42	24112
Prompt 2 (Ablation)	46.99 ± 0.94	98.04 ± 1.36	63.52 ± 0.97	10616	81.56 ± 2.47	33.35 ± 1.59	21565
Prompt 3 (Ablation)	48.29 ± 1.39	97.21 ± 1.15	64.52 ± 1.34	10468	83.16 ± 2.07	34.88 ± 1.98	21268
Prompt 4 (Ablation)	46.12 ± 1.00	98.83 ± 1.30	62.88 ± 1.07	11388	81.75 ± 1.78	39.06 ± 2.30	20479
Prompt 5 (Ablation)	44.55 ± 1.21	99.58 ± 0.78	61.56 ± 1.21	12478	85.22 ± 1.88	49.12 ± 2.00	24336
Prompt 5 (Rubric)	47.36 ± 1.25	89.51 ± 1.74	61.93 ± 1.29	4587	100.00	67.39 ± 2.87	13153
Prompt 6 (Rubric)	46.47 ± 1.18	98.34 ± 1.29	63.11 ± 1.26	9273	99.81 ± 0.57	50.76 ± 2.33	16418
GLM-4.5-Air							
General Summary	20.49 ± 0.56	94.53 ± 1.86	33.68 ± 0.82	16168	62.01 ± 1.92	60.65 ± 2.82	20373
GIMO	54.63 ± 3.16	83.89 ± 3.50	66.13 ± 3.04	11006	70.35 ± 6.45	20.75 ± 2.65	13866
OPC	20.90 ± 0.49	95.89 ± 1.72	34.32 ± 0.71	13331	67.88 ± 2.14	66.02 ± 2.74	19184

D Prompts

OPC Prompt

You are judging the correctness of an LLM−generated proof for a math problem.

Input:

Your input will consist of the following components:
− **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, which was given the following instructions:
<model_prompt>
− 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.
− You should use correct LaTeX notation to write equations and mathematical symbols. You should encompass these equations in appropriate symbols ("\\(" and "\\)" for inline math, "\\[" and "\\]" for block math) to enhance the clarity of your proof. Do not use any unicode characters.
− 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.
</model_prompt>

How the solution should be graded:
A solution should be considered correct even if it would earn 5+/7 points in a standard grading format. Examples of small penalties worth 1 point are if the solution:
− Makes a small computational mistake that can be easily fixed
− Misses an edge case which can be easily proven/disproven
− Skips over a step that follows without much reasoning or manual work
Depending on the severity and the context, you may also not penalise a given error. On the other hand, a solution should be marked as incorrect if:
− It marks a step as trivial, if it is not immediately obvious with little reasoning why this would be the case.
− It omits algebra−heavy computational steps, regardless of whether or not it has outlined the methodology. Skipping shorter computations should be permitted.
− Generalizes over a pattern without rigorously describing the pattern, or without proving any relevant properties.
− It cites a non−existing or unpopular source/Theorem, which cannot be immediately found from searching for it online. Thus, any theorems that can be immediately found and have a Wikipedia article are allowed.

The model has been specifically told that it should not skip steps or mark them as trivial. Any violation of this rule should be considered by assuming the model does not know how to derive the "trivial" step.

Scoring instructions

If you believe the proof is correct, end your analysis with the following two tags:
<summary>One−paragraph summary explaining why the proof is correct</summary>
<judgement>Judgement: Yes</judgement>.

If you believe the proof is incorrect, end your analysis with the following two tags:

<summary>One−paragraph summary explaining why the proof is incorrect</summary>
<judgement>Judgement: No</judgement>.

Problem Statement:
{problem}

Model Solution:
{proof}

General Summary Prompt

[Instructions]

I will provide a math problem along with a solution. Your task is to review each step of the solution in sequence, analyzing, verifying, and critiquing the reasoning in detail. You need to provide the analyses and the conclusion in the following format:

* When you analyze each step, you should use proper verification, recalculation, or reflection to indicate whether it is logically and mathematically valid. Please elaborate on the analysis process carefully.
* If an error is detected in any step, you should describe the nature and cause of the error in detail, and suggest how to correct the error or the correct approach. Once a step is found to contain any error, stop further analysis of subsequent steps and provide your judgement.
* If no error is detected in any step, you should provide your judgement.

[Format]

After your analysis and conclusion, your response MUST follow this exact format:

For correct solutions, you must end your response with:
<summary>One−paragraph summary explaining why the solution is correct</summary>
<judgement>Judgement: Yes</judgement>

For incorrect solutions, you must end your response with:
<summary>One−paragraph summary explaining why the solution is incorrect</summary>
<judgement>Judgement: No</judgement>

[Problem]

{problem}

[Solution]

{proof}

GIMO Prompt

You are an expert mathematician and a meticulous grader for an International Mathematical Olympiad (IMO) level exam. Your primary task is to rigorously verify the provided mathematical solution. A solution is to be judged correct **only if every step is rigorously justified.** A solution that arrives at a correct final answer through flawed reasoning, educated guesses, or with gaps in its arguments must be flagged as incorrect or incomplete.

Instructions

1. Core Instructions
* Your sole task is to find and report all issues in the provided solution. You must act as a **verifier**, NOT a solver. **Do NOT attempt to correct the errors or fill the gaps you find.**

* You must perform a **step-by-step** check of the entire solution. This analysis will be presented in a **Detailed Verification Log**, where you justify your assessment of each step: for correct steps, a brief justification suffices; for steps with errors or gaps, you must provide a detailed explanation.

2. How to Handle Issues in the Solution
When you identify an issue in a step, you MUST first classify it into one of the following two categories and then follow the specified procedure.

* **a. Critical Error:**
  This is any error that breaks the logical chain of the proof. This includes both **logical fallacies** (e.g., claiming that 'A>B, C>D' implies 'A-C>B-D') and **factual errors** (e.g., a calculation error like '2+3=6').
  * **Procedure:**
    * Explain the specific error and state that it **invalidates the current line of reasoning**.
    * Do NOT check any further steps that rely on this error.
    * You MUST, however, scan the rest of the solution to identify and verify any fully independent parts. For example, if a proof is split into multiple cases, an error in one case does not prevent you from checking the other cases.

* **b. Justification Gap:**
  This is for steps where the conclusion may be correct, but the provided argument is incomplete, hand-wavy, or lacks sufficient rigor.
  * **Procedure:**
    * Explain the gap in the justification.
    * State that you will **assume the step's conclusion is true** for the sake of argument.
    * Then, proceed to verify all subsequent steps to check if the remainder of the argument is sound.

3. Output Format
Your response MUST be structured into three XML sections with the tags: <summary>, <detailed_verification>, and <judgement>.

* **a. Summary**
  Wrap this section within <summary>...</summary> tags.
  This section MUST be at the very beginning of your response. It must contain two components:
  * **Final Verdict**: A single, clear sentence declaring the overall validity of the solution. For example: "The solution is correct," "The solution contains a Critical Error and is therefore invalid," or "The solution's approach is viable but contains several Justification Gaps."
  * **List of Findings**: A bulleted list that summarizes **every** issue you discovered. For each finding, you must provide:
    * **Location:** A direct quote of the key phrase or equation where the issue occurs.
    * **Issue:** A brief description of the problem and its classification (**Critical Error** or **Justification Gap**).

* **b. Detailed Verification Log**
  Wrap this section within <detailed_verification>...</detailed_verification> tags.
  Following the summary, provide the full, step-by-step verification log as defined in the Core Instructions. When you refer to a specific part of the solution, **quote the relevant text** to make your reference clear before providing your detailed analysis of that part.

* **c. Judgement**
  This section MUST be at the very end of your response. For correct solutions, you must end your response with EXACTLY:
  <judgement>Judgement: Yes</judgement>
  For incorrect solutions, you must end your response with EXACTLY:
  <judgement>Judgement: No</judgement>

Example of the Required Summary Format
This is a generic example to illustrate the required format. Your findings must be based on the actual solution provided below.

<summary>
Final Verdict: The solution is **invalid** because it contains a Critical Error.

List of Findings:
* **Location:** "By interchanging the limit and the integral, we get..."
  * **Issue:** Justification Gap – The solution interchanges a limit and an integral without providing justification, such as proving uniform convergence.
* **Location:** "From $A > B$ and $C > D$, it follows that $A−C > B−D$"
  * **Issue:** Critical Error – This step is a logical fallacy. Subtracting inequalities in this manner is not a valid mathematical operation.
</summary>

==========
Problem

{problem}

==========
Solution

{proof}

==========
Verification Task Reminder

Your task is to act as an IMO grader. Now, generate the <summary>, the <detailed_verification>, and the <judgement> for the solution above. In your log, justify each correct step and explain in detail any errors or justification gaps you find, as specified in the instructions above. End your response with the required XML tags.