Escaping the Cognitive Well: Efficient Competition Math with Off-the-Shelf Models

Xingyu Dang[*] Princeton University
Rohit Agarwal[*] Princeton University
Rodrigo Porto Princeton University
Anirudh Goyal

Liam Fowl[*†] Princeton Language and Intelligence
Sanjeev Arora[†] Princeton Language and Intelligence
arora@cs.princeton.edu

Abstract

In the past year, custom and unreleased math reasoning models [Google DeepMind, 2025a, Wei et al., 2025] reached gold medal performance on the International Mathematical Olympiad (IMO). Similar performance was then reported using large-scale inference on publicly available models [Luong et al., 2025, Shao et al., 2025] but at prohibitive costs (e.g., $3000 per problem). In this work, we present an inference pipeline that attains best-in-class performance on IMO-style math problems at an average inference cost orders of magnitude below competing methods while using only general-purpose off-the-shelf models. Our method relies on insights about *grader failure* in solver-grader pipelines, which we call the *Cognitive Well* (iterative refinement converging to a wrong solution that the solver as well as the pipeline's *internal* grader consider to be basically correct). Our pipeline addresses these failure modes through *conjecture extraction*, wherein candidate lemmas are isolated from generated solutions and independently verified alongside their negations in a fresh environment (context detachment). On IMO-ProofBench Advanced (PB-Adv), our pipeline achieves 87.6% expert-graded performance using Gemini 3.1 Pro. This surpasses the performance of the DeepThink IMO gold-winning model, and more than triples the success rate of the next best publicly accessible pipeline operating at affordable cost[3].

1 Introduction

Reasoning-based models have shown increasingly impressive progress on mathematical tasks. However, publicly available models cannot reliably solve difficult math problems during a *single call* [Balunović et al., 2025]. Instead, they require a staged solution process, with a candidate solution passing through many rounds of grading, feedback, and refinement. In summer 2025, leading AI companies announced that their customized pipelines with models trained to do "parallel thinking" had achieved gold-level performance at the 2025 International Math Olympiad (IMO) [Google DeepMind, 2025a, Wei et al., 2025]. Soon after, other groups reported similar performance using large-scale agentic pipelines on publicly-available models. For example, Huang and Yang [2025] achieved gold medal performance via multiple parallel runs, each involving around 20 rounds of interaction, with each round in turn involving a "Grader" critiquing the current proof, and the "Solver" refining the proof to address the feedback. In their setup, solving an IMO level math problem may use

[*]Equal contribution.
[†]Equal advising.
[3]DeepSeekMath V2 reaches 61.9% but at an estimated $3000/question, making it inaccessible to most researchers. Among pipelines costing under $300/question, the next best is Huang and Yang [2025] at 24% expert-graded accuracy.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

up to 100 re-runs of the pipeline, which could amount to *thousands* of LLM calls per problem [Luong et al., 2025]. A similar pipeline based on a specialist model by DeepSeek [Shao et al., 2025] reached gold-level performance by allowing up to 64K LLM calls for a maximum budget of 8 billion tokens per problem, which would convert to an estimated 3000 US dollars in API costs. Such pipelines remain too costly for most researchers, let alone enthusiasts.

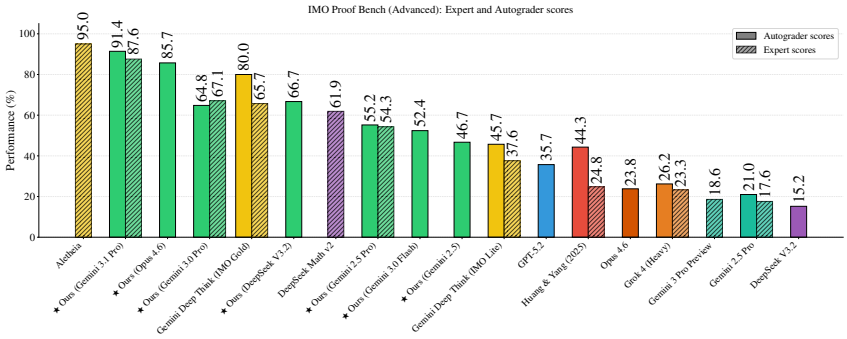

Figure 1: Results for our proposed method versus competing methods and models. Both expert graded and autograded results included (where possible). Our proposed method outperforms all open-source pipelines, while being significantly cheaper than these pipelines (cf. Figure 2) and ranks second only to a custom and unreleased pipeline. Comparison results taken from Luong et al. [2025] where applicable, and from Shao et al. [2025]. Autograder results calculated using the resources published in Luong et al. [2025].

In an effort to measure the rapid progress in this area, a recent paper [Luong et al., 2025] introduces evaluations (and accompanying leaderboards) to measure LLM-based systems' abilities to solve/grade IMO style problems. Of special note is IMO-ProofBench (Advanced) –henceforth called PB-Adv– consisting of 30 curated IMO-style questions that challenge today's models. Proprietary math models Google Deep Think (Lite) and Deep Think (Gold) score 39% and 65% respectively but their compute usage is unreleased. A concurrent work to ours, Aletheia [Luong and Mirrokni, 2026], builds off the newest (unreleased and custom) Gemini Deep Think and achieves state-of-the art performance on PB-Adv, with unspecified cost and compute. Another competing pipeline, DeepSeekMath V2, reaches 61.9% but, as noted, allows up to 3000 USD/question in API cost. The Huang and Yang [2025] pipeline, running on Gemini 2.5 pro with a budget exceeding 300 USD/question, attains only 24% accuracy. Smaller, custom trained models and pipelines offer more affordable options, but do not achieve competitive performance on the more challenging PB-Adv set [LM-Provers et al., 2026]. We introduce a new pipeline that achieves best-in-class performance on a budget that is more than an order of magnitude smaller than comparable methods (cf. Figure 2).

1.1 Identifying Failure Modes

Why do the aforementioned pipelines need expensive parallelism for hard problems? It helps mitigate two phenomena:

Cognitive Plateau: The solver has made some progress, but much of the problem remains unsolved. On successive proofs, the grader points out flaws, but the solver struggles to fix them all, and the grade does not meaningfully change. We call this a *cognitive plateau*. Massive parallelism acts as a form of exploration and discovers rare good ideas, which the grader recognizes as progress.

Cognitive Well: Iterative refinement converges to an imperfect (even dead wrong) solution that has high enough logical consistency that the grader –usually sharing the solver's parameters and sometimes its context window–scores it positively, overlooking the fatal flaws to be "minor slips." For example, Luong et al. [2025] reports that on 28 out of 30 PB-Adv problems, the full Huang and Yang [2025] pipeline produced a solution that received a perfect $7/7$ from the pipeline's internal grader in 5 independent calls, but a majority of times the solution was completely wrong–as judged by the benchmark's grader provided with a reference human solution[4]. We conjecture that such

[4]Our investigation revealed that this arises from the solver exploiting a logical loophole in the grader prompt. This is suggestive of a form of reward hacking.

failure modes are the reason Shao et al. [2025] grades each answer 64x times in parallel. We give a more in-depth discussion of the Huang and Yang [2025] grader in Appendix B.4.

We use "Cognitive Well" for an inference-time dynamic arising from multi-round solver–grader interaction: the solver edits a proof to satisfy a grader sharing its weights. This is related to verifier overconfidence, self-confirmation, and reward hacking; we do not claim that fooling a verifier is new. Operationally, a well is indicated when a high-scoring in-context solution is rejected under independent clean-context grading, or its load-bearing subclaims fail verification after separation from the parent proof.

Appendix D presents examples that illustrate how our method circumvents the above failure modes. The improvements demonstrated in Figures 1 and 2 stem from new conceptual fixes we propose for these issues, as described in Section 2. The importance of the grader role is emphasized in Luong et al. [2025] by their GraderBench evaluation and we report experiments on this evaluation in Section 3.4.

1.2 Misconception: "Grader as compass"

It may appear fraught to allow the same underlying model to produce proofs as well as grade them (via two copies running solver and grader processes). But the intuition supporting this practice is that grading a math proof[5] seems easier than **finding** a correct proof[6], so one expects any given model to be *better* as grader than it is as solver. Thus *in principle* a model should be able to reliably grade (via an independent call with clean context) its own solution.

However, the above intuition only supports **binary** (correct/incorrect) verification. In practice it is tempting to use grader feedback as a *measure of progress*. This is untrustworthy, as the grader may not know how to solve the question. The question may even be functionally impossible to solve in a single model call. Thus, upon detecting gaps in a proof, the grader cannot discern how serious/fixable they are, and thus is unable to estimate the point penalty. Grader scores can fluctuate in counterintuitive ways: a new idea may represent progress for the solver, but may make the grader note new hurdles and reduce its score. The *Cognitive Plateau* arises due to this unreliable signal. The *Cognitive Well* is more counter-intuitive: the grader's near-total acceptance of a deeply flawed solution. We hypothesize that the cause is the multi-step interactive process where the solver improved the proof in response to the grader's feedback. The final flawed proof thus looks largely or completely OK to the computationally-limited grader —this is reminiscent of *reward hacking* and also of stable local minimum in optimization theory. Motivated by this finding, our method limits the grader's feedback to a short list of mistakes, some phrased as questions.

Related Area: Formal Proofs. A parallel research pillar in AI for Math involves models that generate formally verifiable proofs –usually in LEAN [Moura and Ullrich, 2021]–instead of in natural language; see Google DeepMind [2024], Lin et al. [2025a,b], Ren et al. [2025]. For this work, we focus on natural language proving as this represents the forefront for mathematical abilities of current off-the-shelf LLMs.

1.3 Contributions and Paper Organization

The following are our core contributions.

1. A powerful new solution pipeline for competition math problems using only off-the-shelf models. It achieves best-in-class performance and costs up to two orders of magnitude lower than competing methods. To achieve this, we describe new solving methodologies, as well as uncovering insights into current pipelines' failures.
2. Identifying the "Cognitive Wells" trap in solver–grader systems and providing an effective solution for it involving a conjecture-based context-detachment mechanism (prove conjecture and its negation in fresh context) that measurably reduces false-positive wells and boosts performance (cf. Section 2.1).
3. New methods for designing more effective graders for use within a math-solving pipeline, as well as new suggestions for how to evaluate graders (cf. Section 3.4).

[5]That is, checking that each line follows from previous lines and the final step contains the desired conclusion.

[6]The intuition is formalized as the famous $P \neq NP$ conjecture Arora and Barak [2009].

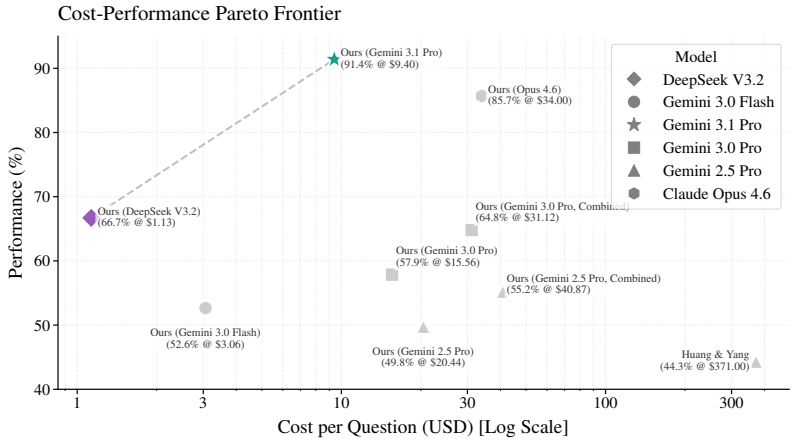

Figure 2: Pareto frontier of cost vs. performance measured using the autograder in Luong et al. [2025]. “Single" denotes that the output of a *single* run of the pipeline. “Combined" denotes our standard algorithm for PB-Adv which uses a further model call to choose from two parallel pipeline runs - cf. Section 2.2 “Post Enhancement and Aggregation"). All costs calculated with January 2026 pricing. The Gemini 3.0 Pro, Gemini 2.5 Pro, and Gemini 3.0 Flash points without a Combined suffix are Single runs. All plotted scores are autograded; the 87.6% headline is expert-graded. The plotted Gemini 3.1 Pro score of 91.4% is Combined, whereas its $9.40 label is the Single-run cost. The Combined headline variant costs roughly twice as much.

4. Open-sourced prompts and methodology —ready to run on public models. We show the efficacy of our pipeline on a range of frontier and open-weight models—Gemini 3.1 Pro, Gemini 3.0 Pro, Gemini 3 Flash, Gemini 2.5 Pro, Claude 4.6 Opus, DeepSeek-V3.2—at an affordable cost; this significantly lowers the barrier to entry for enthusiasts and researchers.

2 Method Overview

Solver-grader pipelines usually involve an **orchestrator** process that dynamically extracts contexts from previous LLM calls and uses them to prompt subsequent calls. We created a *“dialectic based”* prompt-engineering methodology whereby, instead of asking the model to act as, say, an IMO grader, we instruct it to use all provided materials to conduct a multi-round dialectic among some named personas following certain rules. For example, one persona may be extremely picky about slips in the proof, whereas another might be asked to argue that it represents standard practice. Tweaks to each rule changes the final dialectic in fairly predictable ways (see Appendix F) and adding new personas allows new functionalities (e.g., conjecture extraction). In a single model call our dialectic-based prompts seem to perform somewhat better than more conventional prompts (see Appendix B.6 and Table 7), but overall, we stress the main benefit is convenience and precise control[7].

2.1 Pipeline Design Ideas

To address the failure modes described in Section 1.1 while maintaining computational efficiency, our pipeline rests on three core design principles:

1. **Narrow Width (Limited Parallelism):** Brute-force parallelism (e.g., maintaining 64 parallel branches) incurs high costs and makes monitoring difficult. Inspired by recent PDR method [Madaan et al., 2025], we limit parallelism to a modest level (typically $K = 4$ concurrent solutions). The next group of solutions is generated conditioned on the current state, which is the previous group of solutions, feedback from the grader, as well as the results of the conjecturing process described below.

[7]We refer to our pipeline as “Momus pipeline,” with Momus being a picky persona that was named after a mythical figure in Greek philosophy.

2. **Conjecture Extraction:** Pipelines often stall because the internal grader cannot distinguish between a "minor slip" and a fatal gap. To resolve this, we employ a *Solver with Conjectures* prompt. This module is given the current set of stalled proofs and asked to explicitly formalize logical gaps as self-contained conjectures (and also formalize their negations). This forces the model to define exactly what is missing to complete a proof, transforming vague uncertainty into falsifiable mathematical statements.
3. **Contextual Detachment:** The "Cognitive Well" occurs when a grader becomes "contaminated" by the context of a plausible-but-wrong proof. Combating this requires stripping extracted conjectures from their parent context, and feeding them –as well as their negations– as new problems to fresh solver instances. If that results in proving the *negation* of a necessary conjecture, including this as "additional material" for the solver in the next step can nudge it to discredit the ideas of the stalled proof, and think anew.

2.2 Pipeline Architecture

Our inference pipeline organizes these principles into a stateful loop. In previous pipelines the state is a single candidate solution –or a few solutions. Our conjecture process turns the state into a set of candidate proofs for the original problem, grader feedback, as well as proofs/disproofs for intermediate conjectures. The process has three phases:

Phase I: Limited Exploration
We generate a small set of candidate solutions (typically $K = 4$) using the Dialectic Solver.

1. **Generation:** The model produces candidate proofs $P_1 \dots P_K$.
2. **Adversarial Grading:** A separate instance of the model grades each P_i. The grader is prompted to use a "guilty until proven innocent" framework for listing specific logical gaps.
3. **Outcome:** If a solution receives a perfect score ($7/7$) across three independent grading runs, it is accepted. If not, the pipeline enters Phase II.

Phase II: Contextual Detachment
This phase targets the *Cognitive Plateau* and *Cognitive Well*. When solutions are plausible but unverified (or stuck at a score $\leq 6/7$), we assume the verifier is "contaminated" by the context of the incorrect proof and execute the following:

1. **Extraction:** We prompt the model to identify load-bearing conjectures $\{C_1, \dots, C_m\}$ from top-scored candidate proofs. These are statements which, if true, would help complete the proof.
2. **Isolation:** Each conjecture C_j is stripped of its parent proof context as an independent problem.
3. **Bisection:** For each conjecture, we make one independent DIALECTICSOLVE call for C_j and one for $\neg C_j$, each scored by the Momus grader. The Phase-I outer refinement loop is not repeated for either side.
4. **Resolution:** A direction is admitted only if its Momus grade is at least $6/7$ and the opposite direction scores below $6/7$. Grade-7 proofs enter $\mathcal{M}_{lemma}$ as usable lemmas; grade-6 partial proofs are retained separately as labeled reference material. If both directions pass, neither is admitted.

Phase III: Refinement with Global Memory
The pipeline launches a new round of solvers. However, all but one of these solvers are no longer zero-shot. They are injected with the "Global Memory" $\mathcal{M}_{lemma}$ containing the independently verified lemmas from Phase II. If a positive conjecture was proven, this allows the new solver to make further progress on the suggested proof ideas. If a negative conjecture was proven, this allows us to potentially escape a Cognitive Plateau and change ideas. We also provide grade-6 partial proofs from the conjecture round as labeled reference material, as this may be helpful to a solver. Attempts below grade 6 are not passed to the final solve. The one zero-shot solver remains to keep diversity and avoid falling into a Cognitive Well.

Post Enhancement and Aggregation. If the conjecture iteration budget is exhausted without yielding a verified solution, the pipeline executes a final post-enhancement step. Here, we run 2 parallel

sessions of Phase II, where the conjecture extractor is rewriting the proof based on the conjectures. We pass grade-7 lemmas as usable proofs and grade-6 partial proofs as labeled reference material to Phase III for the final solving; attempts below grade 6 are not passed on. Algorithm 1 uses the same two-session extraction and verification step.

To maximize performance, we may run two instances of the entire pipeline in parallel. If so, a final "Judge" combiner persona compares the final outputs of both pipelines. We employ this strategy for more difficult questions, but opt for a more efficient single pipeline run for simpler problems (cf. Figure 5).

For a more formal description of our algorithm, please refer to Appendix Section A, and specifically Algorithm 1. Our pipeline is generally model agnostic - although performance does scale with base model capability. We provide ablations for important parts of our pipeline in Section 3 and in the Appendix.

We also provide high-level visualization of this pipeline architecture in Appendix Figure 4.

3 Results

IMO-Bench represents the most natural domain in which to test our methods. While the difficult subset PB-Adv comprises only 30 questions, it is designed to be difficult for current systems and relatively contamination-free.

IMO ProofBench judges correctness using the full proof/reasoning, rather than just a numeric final answer/expression. It comes with an autograder prompt provided with detailed gold solutions and scoring rubrics Luong et al. [2025]. Its ratings correlate strongly with human judgments ($r = 0.93$; see Section 3.4) but can be less accurate on AI generations. Expert human grading is thus the gold standard.

Because of limited resources and the need to compare to results previously reported, our headline numbers come from four Gemini models: Gemini 3.1 Pro, Gemini 3.0 Pro, Gemini 3.0 Flash, and Gemini 2.5 Pro. However, we additionally report results with further backbones (Claude 4.6 Opus, the open-weight DeepSeek-V3.2) in Section 3.1.

While most of our reported numbers correspond to the autograder, we also used human graders (IMO medalists, math competition problem writers, and mathematicians) for one full run each of Gemini 2.5 Pro, Gemini 3.0 Pro, and Gemini 3.1 Pro (30 problems per backbone). Appendix B.9 gives the blind grading protocol, coverage, and expert–autograder agreement. We found high consistency between these (blind) expert scores and autograder scores, which suggests our pipeline may avoid discrepancies between the two grading methods present in other works (e.g. see Figure 1 for the performance of DeepThink models and HY pipeline as reported in Luong et al. [2025]). We think that divergence between the Autograder and human graders should be evaluated as possible indications of in-context reward hacking mentioned in Section 1.1.

In Figure 1 we present a comparison of our method's performance on IMO Proof Bench (Advanced) to a wide range of other frontier models and pipelines using both autograder performance and expert grader performance (where possible). According to the expert human grading, our pipeline scores second - only to the concurrent (and unreleased) Aletheia model from Google DeepMind that was announced in February 2026. Notably, our best run outperforms the earlier Deep Think IMO-Gold model, and the massively parallel (and expensive) pipeline incorporating the specialst DeepSeek Math v2 model [Shao et al., 2025]. Also of particular note is our performance compared to the Huang and Yang [2025] pipeline where, when using the same exact base model, we more than double the performance (according to expert grading) while operating at a budget an order-of-magnitude lower.

3.1 Generalization Across Backbones

The Gemini family powers both the leading academic pipeline of Huang and Yang [2025] and the 2025 IMO-gold winning DeepThink system, which is why our headline numbers use it. To verify that the gains are not specific to Gemini, we additionally ran the pipeline with Claude 4.6 Opus, the open-weight DeepSeek-V3.2 as the backbone for all roles (solver, grader, and conjecture-extractor). Appendix Table 4 reports the autograded average score on PB-Advanced for each backbone alongside the score of a single base call to the same model.

Across all three additional backbones our pipeline drives substantial gains. The largest is on Claude 4.6 Opus, which jumps from a 23.8% base call to 85.7%; the open-weight DeepSeek-V3.2 jumps from 15.2% to 66.7%. These results indicate that the gains attributed to conjecture extraction and contextual detachment are not artefacts of a single model family.

3.2 Performance on Additional Benchmarks

PB-Adv is the gold standard for difficult competition level mathematics (for example, this benchmark serves as the main barometer for Aletheia, and DeepSeek Math v2). However, PB-Adv comprises only 30 questions, and therefore is a relatively narrow basis for evaluation. Thus, we additionally ran our pipeline (with Gemini 3.0 Flash, our cheapest backbone) on two larger benchmarks: **ProofBench** (145 USAMO/IMO/Putnam questions, distinct from IMO-ProofBench) and **IMO-AnswerBench** (400 questions, final-answer evaluation). Baselines for both benchmarks are taken from QED-Nano [LM-Provers et al., 2026]. Appendix Table 3 reports the results. Our pipeline outperforms all baselines on both benchmarks despite using our weakest backbone. Note that broader evaluations remain cost-prohibitive at academic budgets: running our pipeline on the 400 IMO-AnswerBench problems with Gemini 3.0 Flash alone costs roughly $\$1{,}000$.

3.3 Solver Prompt Evaluation

To isolate the source of our performance gains, we examined whether our internal dialectic (Momus) prompting strategy provides an inherent advantage over standard prompting. We conduct a controlled comparison against the solver prompt from Huang and Yang [2025] on PB-Basic (30 problems, 4 independent runs each).

We observed a modest performance difference between the two strategies: the Momus solver achieved an average-of-4 score of 5.10/7, while the Huang-Yang solver got 4.92/7. Detailed results are available in Appendix B.6. Overall, the dialectic approach does not seem to yield a significantly better solver. This parity suggests that the significant performance difference observed in Figure 1 stems primarily from our pipeline's other innovations—specifically, grading, conjecture extraction and contextual detachment—rather than the initial solver prompt formulation. Our dialectic approach was used for prompt design for these other roles as well.

3.4 Grader Evaluation

Because the internal grader in a pipeline is of paramount importance for the success of the solver, we additionally conduct evaluations on the Graderbench benchmark in Luong et al. [2025]. Graderbench evaluates *grader performance* using a dataset of AI-generated solutions to PB-Adv questions with accompanying human grades on 0-7 scale. **Accuracy** is measured via a 4-way classification task with labels $0/1/6/7$: label 1 standing for $\{1, 2, 3\}$ and 6 for $\{4, 5, 6\}$, since that aligns with human grading practices. In this section we use it to evaluate our grader and also to shed light on ways in which the benchmark may need rethinking.

In Section 1.2, we cautioned against taking the internal grader's numerical score seriously, since accurate scoring requires estimating what fraction of the problem is unsolved. This is infeasible given the grader's compute limitations, and no access to a reference solution/rubric. The numerical grade is not shown to the model as a refinement signal: refinement uses the written feedback and list of slips. The orchestrator does use grades to rank candidates, select conjecture-extraction seeds and guided-solving contexts, admit conjectures, choose the final output, and exit early upon a perfect score (with zero slips) thrice in a row. Our **Momus grader** is designed with Cognitive Wells in mind: it prioritizes **strictness**.

It is still informative to see evaluations on GraderBench for: (i) Our vanilla **Momus Grader** that is used in our solution pipeline, (ii) a variant **Momus Grader w/** 3 **Pipeline Solutions** inspired by the above-mentioned argument of Section 1.2. To give our Momus grader the benefit of more compute, we provide it with three sample solutions (from evenly spaced runs) generated during our pipeline's attempt to solve the same problem. Even though our pipeline fails to solve many problems, one hopes that the intermediate attempts nevertheless help the grader understand the problem's intricacies.

GraderBench evaluation centers on (i) **Accuracy (Acc)** on the classification task, and (ii) **Mean Absolute Error (MAE)**: As in Luong et al. [2025], model-predicted scores are rounded to $0/1/6/7$, and compared with human grades on a 7-point scale.

All our experiments in this section used Gemini 2.5 Pro. Numbers in the table for other models are taken from Luong et al. [2025] which in turn evaluated using the **ProofAutoGrader** prompt without the reference human solution.

Table 1: Comparison of Momus Prompts vs. Other models on GraderBench (1000 datapoints). No reference solution provided. Non-Momus models are run with the ProofAutoGrader prompt. MAE is expressed as a percentage of the maximum score of 7.

Method	**Acc** $\uparrow$	**MAE** $\downarrow$
Gemini 2.5 Pro	44.3%	30.2%
Momus	28.1%	25.9%
o4-mini (high reasoning)	47.3%	25.2%
Gemini 2.5 Deep Think	52.5%	20.5%
o3	54.0%	20.2%
Momus w/ p. sols	49.8%	19.9%
Gemini 2.5 Deep Think (IMO Gold)	50.2%	18.4%

Table 1 shows that Momus with pipeline solutions is competitive with the best graders.

Does GraderBench performance capture usefulness?

We hypothesize that standalone grading accuracy need not predict a grader's usefulness within a solution pipeline. Table 1 motivates this distinction, but does not establish that vanilla Momus produces better end-to-end results than the pipeline-solutions variant. The token-matched grader swap in Appendix B.11 favors Momus over the vanilla grader directionally, but is not statistically significant.

Pipeline performance (especially when using low parallelism) depends closely on the following metrics, with *positive* considered a score of 6 or higher; a *negative* considered a score of 5 or lower[8].

- **False Positive Rate (FPR)**: $P(\text{pred} \geq 6 \mid \text{human} \leq 5)$. In a pipeline setting this is a critical failure mode, a form of "Cognitive Well" with increased risk of submitting an incorrect solution.
- **False Negative Rate (FNR)**: $P(\text{pred} \leq 5 \mid \text{human} \geq 6)$. Grader being over-strict is not a critical failure; it triggers another round of exploration that often improves the solution.

The above metrics are shown for a simplified presentation in Table 2; the full picture involves confusion matrices in Figure 7.

Table 2: Comparison of Momus Prompts vs. ProofAutoGrader (with reference solution) on GraderBench (1000 datapoints), all using Gemini 2.5 Pro. FPR columns condition on human grades ≤ 5; the thresholds are $\text{pred} \geq 6$ and $\text{pred} = 7$. Rates are for single calls. MAE is expressed as a percentage of the maximum score of 7.

Method	**FPR** ≥ 6	**FPR** $= 7$	**FNR**	**Acc**	**MAE**
Momus	17.1%	2.7%	26.1%	28.1%	25.9%
ProofAutoGrader	31.6%	21.3%	6.2%	59.4%	17.6%
Momus w/ p.sols	26.9%	21.4%	8.8%	49.8%	19.9%

Momus grader's lower FPR rate reflects an emphasis on a safety margin: being picky helps mitigate Cognitive Wells. Pickiness increases FNR, and the pipeline returns its best solution according to the internal grader. This protection is conditional on the grader's discrimination: it need not retain the best solution under external grading. In fact, many solutions considered imperfect by Momus receive higher scores from the ProofAutoGrader.

[8] If we combine the 0 and 1-3 buckets, then the classification accuracies in Table 2 become more comparable. ProofAutoGrader is 73.5% and Momus with pipeline solutions is 72.0%, while base Momus gets 53.7%.

This raises the crucial point: even the gold ProofAutoGrader might not be best for use in a solution pipeline. It prioritizes matching human grades on the benchmark which rarely fall between 2 to 5. Its higher grading accuracy comes at the expense of worse FPR; whether this degrades end-to-end accuracy when used to terminate a solution pipeline remains a hypothesis. A good middle ground is the **Momus grader with pipeline solutions** which recovers much of the accuracy difference and almost matches the MAE for ProofAutoGrader. However, its FPR is somewhat higher than vanilla Momus; this motivates the hypothesis that it may be less useful inside the pipeline, rather than establishing that conclusion. In summary, a grader's value as a standalone scorer (e.g., as measured by GraderBench) may not be aligned with its role as a source of feedback in a larger pipeline. This tension—and the need for better metrics to capture it—remains an important area for future study.

Momus's single-call $\mathrm{pred} = 7$ false-positive rate is $18/659 = 2.7\%$. Requiring zero slips and three successful verifications further restricts acceptance; these calls may have correlated errors, so we do not cube the single-call rate. The admitted-lemma audit is reported separately in Appendix B.10.

3.5 Conjecture Removal: Analysis

In order to investigate the added value of conjecture extraction, we isolate problems where the Gemini 2.5 Pro version of the pipeline is successful *and* proved at least one conjecture. We then try omitting the conjecture stage entirely. To perform this ablation, we modify our method so that our pipeline simply loops over the parallel solving stage with the standard 4 solvers and the usual Momus grader. Furthermore, we match token counts: that is, if our pipeline uses T tokens, we allow the modified pipeline to also use up to T tokens (early termination is allowed). We then select 5 evenly spaced checkpoints throughout this pipeline's run and grade them with the ProofAutoGrader as well as the internal pipeline grader.

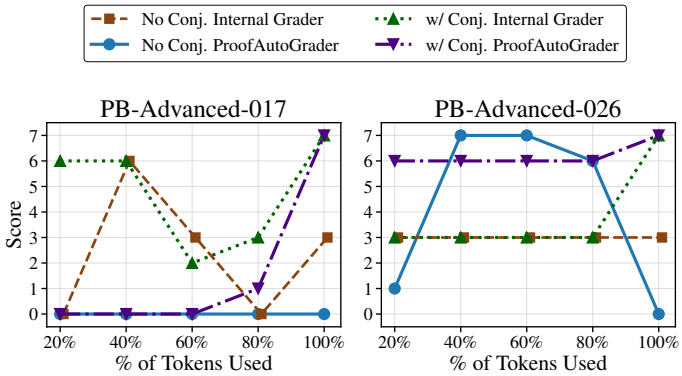

Figure 3: Case study of benefits of conjecture extraction/resolution on the solution pipeline. We run (a) the pipeline without any conjecture steps and (b) with conjecture steps. For both runs, we equalize token budgets. We sample solutions for five phases throughout the runs. We then grade these runs using both the pipeline's internal grader and the external ProofAutoGrader. We see that with conjectures, the ProofAutoGrader scores monotonically increase, while those without may jump wildly or flatline. Spikes in conjecture-based solving correspond to proof/disproof of conjectures.

Figure 3 shows the results on two representative problems. (Appendix B.5 has more examples.) In particular on PB-Adv-017, we see that neither setup makes progress with respect to ProofAutoGrader grades for the first 60% of the run. In the last 40%, proven conjectures accumulate and allow the model to score a perfect 7/7. Contrast this with progress after removal of proved conjectures–the pipeline grader awards 6/7 to a solution that is completely incorrect. This gets it stuck in the "Cognitive Well," unable to make any better solutions. Appendix B.5, includes results on a broader class of problems, exhibiting how inclusion of conjecture extraction reliably stabilizes and increases final scores.

PB-Adv-026 in Figure 3 illustrates a failure of score-driven selection: without conjectures, internal grades do not distinguish a correct solution from incorrect ones, and the run ends at 0 after finding an externally graded 7. This and the other five runs ending at 0 in Appendix B.5 are the conjecture-removed condition; all six score 7/7 with conjectures. The protection remains conditional on

conjecture resolution. The roughly 41% adjudication rate in Appendix B.8 is per conjecture, not per run, which may extract multiple conjectures across rounds.

Aggregate effect at matched compute. The per-problem trajectories above leave open whether the conjecture-removed pipeline could simply "catch up" given enough additional sampling. To rule this out, we ran a token-matched ablation on Gemini 2.5 Pro across all of PB-Advanced. We removed the conjecture-extraction stage entirely and scaled up the standard solver–verifier loop until each problem received up to ~ 2.1M compute tokens—the per-problem budget our full pipeline uses on its hardest PB-Advanced problem. Even with this generous budget, the no-conjecture variant scored only 34.3% on PB-Advanced, compared to 55% for the full pipeline on the same backbone. The gap supports the conjecture stage as a whole at matched compute. Because the ablation removes the entire stage, it does not isolate detachment from extraction and bisection.

4 Conclusion

Our work suggests that reliable LLM theorem proving may not require massive parallelism and thinking budgets. Our efficient inference pipeline enables off-the-shelf models to reach similar performance at a fraction of the cost of comparable published methods. Central to our contribution is the identification of *Cognitive Plateaus* and *Cognitive Wells* wherein pipelines can become trapped in apparently plausible but ultimately incorrect reasoning paths that fool their own internal verifiers. We suspect such failure modes are deep-rooted in current solver/grader pipelines, and may extend to broader AI-for-Science systems. This phenomenon requires careful attention, and further exploration is left for future work, as is better grader evaluation.

We validate our pipeline on the recently released suite of Proof-Bench evaluations. On IMO Proof Bench-Advanced we scored (at the time of the experiments) ahead of all but one competing methods, both released and unreleased, with performance verified by expert graders. Additionally, we probe the role of graders as standalone scorers versus feedback providers and analyze where the two roles may not align.

Acknowledgments and Disclosure of Funding

We thank IMO Medalists and Math Enthusiasts Minjae Kwon, Xinmiao Han, Liangjun Zhong, Geoffrey Sangston and Kaiyue Wen for their help human-grading many of the model-generated proofs. We acknowledge API access from Google Academic Project award. This work was funded in part by ONR, Darpa AIQ (AI-Quantified), Schmidt Foundation, and Google DeepMind.

References

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark*, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Google DeepMind. AI achieves silver-medal standard solving International Mathematical Olympiad problems, July 2024. URL https://deepmind.google/blog/ai-solves-imo-problems-at-silver-medal-level/.

Google DeepMind. Advanced version of Gemini with Deep Think officially achieves gold medal standard at the International Mathematical Olympiad, January 2025a. URL https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/.

Google DeepMind. A new era of intelligence with gemini 3. https://blog.google/products-and-platforms/products/gemini/gemini-3/, 2025b.

Yichen Huang and Lin F Yang. Winning gold at IMO 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025a.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025b.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

LM-Provers, Yuxiao Qu, Amrith Setlur, Jasper Dekoninck, Edward Beeching, Jia Li, Ian Wu, Lewis Tunstall, and Aviral Kumar. Qed-nano: Teaching a tiny model to prove hard theorems. https://huggingface.co/spaces/lm-provers/qed-nano-blogpost, 2026. Blog post.

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

Thang Luong and Vahab Mirrokni. Accelerating mathematical and scientific discovery with Gemini Deep Think. https://deepmind.google/blog/accelerating-mathematical-and-scientific-discovery-with-gemini-deep-think/, February 2026. Google DeepMind Blog.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: LLMs as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pages 625–635. Springer, 2021.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Alexander Wei, Noam Brown, and Sheryl Hsu. OpenAI achievement of gold medal standard at the 2025 International Mathematical Olympiad, July 2025. URL https://x.com/alexwei_/status/1946477753194484181.

A Algorithm Details

Here we provide a high-level schematic of our pipeline, and detailed algorithms.

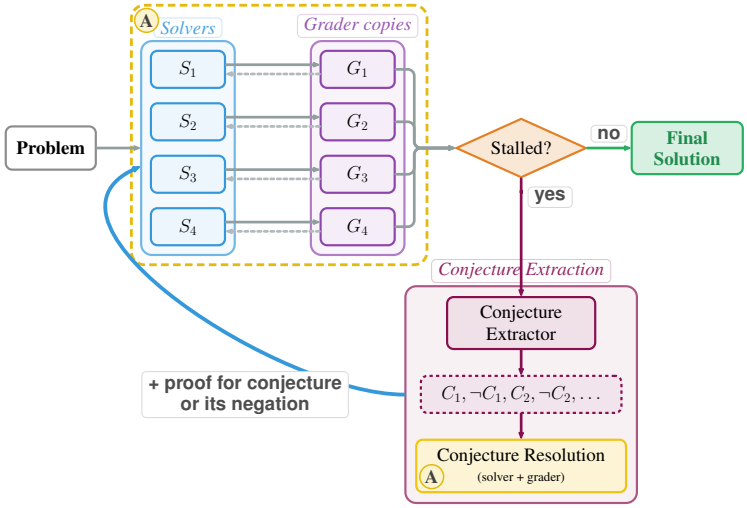

Figure 4: A high level overview of our pipeline. A problem is processed by parallel solver–grader pairs (module **A**). When progress stalls, the *Conjecture Extractor* identifies candidate lemmas C_i and their negations from current proof attempts. Each conjecture is resolved by an independent solver–grader module (also **A**), with proofs (for either the conjecture or its negation) fed back to enable further progress toward the final solution. For conjecture resolution, each side uses one DIALECTICSOLVE call and its Momus grade, not the Phase-I outer loop.

Algorithm 1 is our main pipeline algorithm. Sub-functions are defined in other algorithms. EVAL extracts the Momus grade returned with a DIALECTICSOLVE result; each conjecture side uses one such call, not the Phase-I outer loop. VERIFIEDSUCCESS requires three $7/7$ verifications with zero slips. Grade-7 lemmas and grade-6 reference partials are labeled separately; failed attempts are retained for extraction feedback but are not supplied to the final solve. The configuration value `conjecture_grade_threshold=5` is equivalent to admission at 6 or 7 because the rubric never awards 5.

There are further details on sub-functions in Algorithm 2. Note that Algorithm 2 is orchestrator-level pseudocode: the parallel threads and the separate LAZYPHRASING model call (Gemini 2.5 Flash) represent distinct API calls managed by the Python orchestrator, not steps within a single prompt execution.

Algorithm 1 Memory-Augmented Dialectic Reasoning (Main Pipeline)

Input: Problem P, Initial Iterations $L_0 = 2$, Conjecture Iterations L, Solvers K, Lemma Gate $\tau_\ell = 6$, Final Gate $\tau_f = 7$
Initialize: Solution Memory $\mathcal{M}_{sol} \leftarrow \emptyset$, Lemma Memory $\mathcal{M}_{lemma} \leftarrow \emptyset$, Reference Partials $\mathcal{M}_{ref} \leftarrow \emptyset$, Failure Context $\mathcal{C}_{fail} \leftarrow \emptyset$
// PHASE 1: BASELINE EXPLORATION WITH FEEDBACK REFINEMENT
$\mathcal{S}_{base} \leftarrow \emptyset$
for $i = 1$ **to** L_0 **do**
  for $k = 1$ **to** $K - 1$ **do**
    $Ctx_k \leftarrow \text{SELECTKTHTOP}(\mathcal{S}_{base}, k)$
  end for
  $\mathcal{S}_{base} \leftarrow \mathcal{S}_{base} \cup \text{DIALECTICSOLVE}(P, \emptyset) \cup \bigcup_{k \le K-1} \text{DIALECTICSOLVE}(P, Ctx_k)$
  if $\text{VERIFIEDSUCCESS}(\mathcal{S}_{base}, \tau_f, N = 3)$ **then**
    return $\text{BEST}(\mathcal{S}_{base})$
  end if
end for
$\mathcal{M}_{sol} \leftarrow \text{UPDATE}(\mathcal{M}_{sol}, \mathcal{S}_{base})$
for $l = 1$ **to** L **do**
  // PHASE 2: LEARNING FROM FAILURE
  $S_{seeds} \leftarrow \text{SELECTTOP}(\mathcal{M}_{sol}, N = 2)$
  $\mathcal{H} \leftarrow \text{EXTRACTHYPOTHESES}(S_{seeds}, \mathcal{M}_{lemma}, \mathcal{C}_{fail})$
  $\mathcal{L}_{new}, \mathcal{R}_{new}, \mathcal{F}_{new} \leftarrow \text{VERIFYHYPOTHESES}(\mathcal{H}, \tau_\ell)$
  $\mathcal{M}_{lemma} \leftarrow \mathcal{M}_{lemma} \cup \mathcal{L}_{new}$
  $\mathcal{M}_{ref} \leftarrow \mathcal{M}_{ref} \cup \mathcal{R}_{new}$
  $\mathcal{C}_{fail} \leftarrow \mathcal{C}_{fail} \cup \mathcal{F}_{new}$
  // PHASE 3: SOLVING WITH ENHANCED CONTEXT
  $Ctx \leftarrow \mathcal{M}_{lemma} \cup \mathcal{M}_{ref} \cup \text{RETRIEVEBEST}(\mathcal{M}_{sol})$
  $\mathcal{S}_{guided} \leftarrow \text{DIALECTICSOLVE}(P, Ctx, K - 1)$
  $\mathcal{S}_{fresh} \leftarrow \text{DIALECTICSOLVE}(P, \emptyset, 1)$ {Diversity check}
  $\mathcal{S}_{final} \leftarrow \mathcal{S}_{guided} \cup \mathcal{S}_{fresh}$
  $\mathcal{M}_{sol} \leftarrow \text{UPDATE}(\mathcal{M}_{sol}, \mathcal{S}_{final})$
  if $\text{VERIFIEDSUCCESS}(\mathcal{S}_{final}, \tau_f, N = 3)$ **then**
    return $\text{BEST}(\mathcal{S}_{final})$
  end if
end for
// PHASE 4: POST-ENHANCEMENT FOR NEAR-PERFECT SOLUTIONS
$S^* \leftarrow \text{BEST}(\mathcal{M}_{sol})$
$\mathcal{G}_{check} \leftarrow \text{GRADEINDEPENDENT}(S^*, N = 3)$
if not $\text{ALLPERFECT}(\mathcal{G}_{check})$ **then**
  for $j = 1, 2$ **in parallel do**
    $\mathcal{H}^j_{gaps} \leftarrow \text{EXTRACTGAPS}(S^*, \mathcal{G}_{check})$
    $\mathcal{L}^j_{fix}, \mathcal{R}^j_{fix}, \mathcal{F}^j_{fix} \leftarrow \text{VERIFYHYPOTHESES}(\mathcal{H}^j_{gaps}, \tau_\ell)$
  end for
  $Ctx_{fix} \leftarrow \mathcal{M}_{lemma} \cup \mathcal{M}_{ref} \cup \bigcup_j (\mathcal{L}^j_{fix} \cup \mathcal{R}^j_{fix})$
  $S_{better} \leftarrow \text{DIALECTICSOLVE}(P, Ctx_{fix}, K)$
  $\mathcal{G}_{better} \leftarrow \text{GRADEINDEPENDENT}(S_{better}, N = 3)$
  return $\arg\max_{S \in \{S^*, S_{better}\}} \text{MEAN}(\mathcal{G}_S)$
end if
return S^*

Algorithm 2 Function: DialecticSolve

Input: Problem P, Context Ctx, Count $N = 1$
$\mathcal{S}_{out} \leftarrow \emptyset$
for $i = 1$ **to** N **in parallel do**
  // STEP 1: DRAFT & CHECK
  $S_i \leftarrow \text{LLM}_{\text{solve}}(P, Ctx)$
  if LAZYPHRASING(S_i) **then**
    $S_i \leftarrow \text{LLM}_{\text{solve}}(P, Ctx,$
      feedback="Derive explicitly")
  end if
  // STEP 2: INQUISITORIAL GRADING
  $F_i \leftarrow \text{LLM}_{\text{grader}}(S_i)$
  $S_i' \leftarrow \text{LLM}_{\text{refine}}(S_i, F_i)$
  // STEP 3: RE-GRADE REFINED SOLUTION
  $F_i' \leftarrow \text{LLM}_{\text{grader}}(S_i')$
  $\mathcal{S}_{out} \leftarrow \mathcal{S}_{out} \cup \{(S_i', F_i')\}$
end for
return $\mathcal{S}_{out}$

Algorithm 3 Function: VerifyHypotheses

Input: Candidates $\mathcal{H}$, Lemma Gate $\tau_\ell = 6$
$\mathcal{L}_{proven}, \mathcal{R}_{partial}, \mathcal{F}_{failed} \leftarrow \emptyset, \emptyset, \emptyset$
for all $(C, \neg C) \in \mathcal{H}$ **do**
  $S_{pos} \leftarrow$ DIALECTICSOLVE$(C, \emptyset, 1)$
  $S_{neg} \leftarrow$ DIALECTICSOLVE$(\neg C, \emptyset, 1)$
  $g_{pos} \leftarrow$ EVAL$(S_{pos}), \quad g_{neg} \leftarrow$ EVAL(S_{neg})
  if $g_{pos} \geq \tau_\ell \wedge g_{neg} < \tau_\ell$ **then**
    $(D, S, g) \leftarrow (C, S_{pos}, g_{pos})$
  else if $g_{neg} \geq \tau_\ell \wedge g_{pos} < \tau_\ell$ **then**
    $(D, S, g) \leftarrow (\neg C, S_{neg}, g_{neg})$
  else
    $\mathcal{F}_{failed} \leftarrow \mathcal{F}_{failed} \cup \{(C, \text{Unresolved})\}$
    continue
  end if
  if $g = 7$ **then**
    $\mathcal{L}_{proven} \leftarrow \mathcal{L}_{proven} \cup \{(D, S)\}$
  else
    $\mathcal{R}_{partial} \leftarrow \mathcal{R}_{partial} \cup \{(D, S, \text{Reference only})\}$
  end if
end for
return $\mathcal{L}_{proven}, \mathcal{R}_{partial}, \mathcal{F}_{failed}$

Algorithm 4 Parallel Runs

Input: Problem P, Number Parallel Generations $N = 2$
for $i = 1$ **to** N **in parallel do**
  $S_i \leftarrow$ MEMORYAUGMENTEDDIALECTIC(P)
end for
return PICKBEST$(S_1, \ldots, S_N)$

B Ablations

In this section we track various ablations over benchmarks, graders, and hyperparameters among others.

B.1 Other Benchmarks

While PB-Adv is the most challenging area to measure the performance of models, PB-Basic also presents useful signal for practitioners seeking to evaluate capabilities on slightly less challenging problems. We demonstrate the success of our model on PB-Basic in Figure 5. We see that our pipeline – run only once using several architectural variants, achieves competitive scores with the top pipelines. Since it is allowed to converge early, we incur minimal inference costs, for example 4.54 USD/question for 3.0 Pro, and 1.23 USD/question for 3.0 flash.

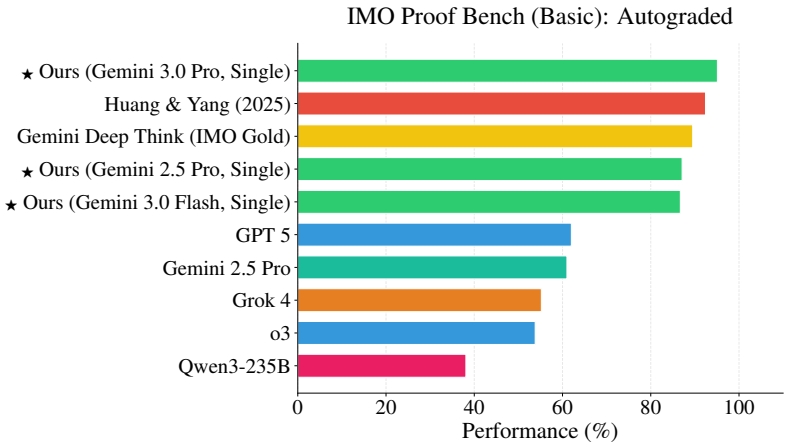

Figure 5: Results for IMO Proof Bench (Basic) according to the IMO-Bench autograder. “Single" here denotes that the output of a *single* run of the pipeline was submitted (as opposed to our standard algorithm for PB-Adv which uses a further model call to choose from two parallel pipeline runs - cf. Section 2.2 “Post Enhancement and Aggregation"). All comparison results taken from Luong et al. [2025].

In addition, we report performance on two more benchmarks: ProofBench - a similarly named but independent benchmark of roughly 400 questions from a variety of sources including the Putnam exam, and IMOAnswerBench - a final answer only benchmark comprising a larger number of competition math questions.

Table 3: Results on ProofBench and IMO-AnswerBench. Our pipeline runs on Gemini 3.0 Flash; baselines from QED-Nano.

Model	**ProofBench**	**IMO-AnswerBench**
Qwen3-30B-A3B-Thinking-2507	26.1	67.0
Qwen3-235B-A22B-Thinking-2507	33.7	70.5
GPT-OSS-120B	47.5	70.5
DeepSeek-Math-V2	60.6	75.8
Gemini 3 Pro	66.7	83.2
Ours (Gemini 3.0 Flash)	**80.3**	**91.3**

Our pipeline outperforms all baselines on both benchmarks despite using our weakest backbone. Note that broader evaluations remain cost-prohibitive at academic budgets: running our pipeline on the 400 IMO-AnswerBench problems with Flash alone costs roughly $\$1{,}000$.

B.2 Backbone Ablation

Here we show gains on PB-Adv using different backbone models for each of our pipeline roles.

Table 4: Generalization across backbones on PB-Advanced. "Base Avg" is the autograded average score (%) for a single call to the model; "Ours Avg" is the score of our pipeline using that model as the backbone. "Graded (Ours)" is the number of problems for which grading had completed at the time of writing.

Backbone	**Base Avg (%)**	**Ours Avg (%)**
Claude 4.6 Opus	23.8	85.7
DeepSeek-V3.2	15.2	66.7

B.3 Statistical Significance

Significance testing in this setting is challenging because two sources of randomness are at play: prover sampling and LLM-grader sampling. Hiring expert human graders at scale to mitigate the latter is financially infeasible at academic budgets. To bound prover variance we ran 10 independent trials on Gemini 3.0 Flash—our cheapest backbone—on PB-Advanced; running the same number of trials on Gemini 2.5 Pro would have cost over $\$12{,}000$. These trials yield a Student-t 95% confidence interval on the mean autograded score of $[49.1\%, 56.1\%]$ on PB-Advanced. Even the lower bound exceeds the autograded score reported by Huang and Yang [2025], despite our pipeline using a weaker backbone and a budget orders of magnitude smaller.

B.4 Huang and Yang [2025] Grader Analysis

Due to resource constraints, we were unable to run the expensive Huang and Yang [2025] pipeline more than a few times on PB-Adv. We focus on a single run, using the default settings and graded using ProofAutoGrader, which does not reveal all of the false positives that Luong et al. [2025] observe. The results of grading this run is summarized in Table 5.

Table 5: The counts of how many questions received each possible autograder result on Huang and Yang [2025] pipeline solutions. None of the proposed solutions earned a 1 through 6. Note that the pipeline does not finish (DNF) if it does not grade a solution correct 5 times in a row.

ProofAutoGrader Score	**Count**
0/7	1
7/7	14
DNF	15

The only false positive we observed in this run is **PB-Adv-011**. For completeness, the full solution text is in Appendix G.1.

We study this case further in Table 6 by running 20 independent verifications with the Huang and Yang [2025] verifier on the following solutions:

- The solution output by the Huang and Yang [2025] pipeline (that was verified 5 times in a row).
- The solution output by the pipeline, without the following lines: "I have not found a complete solution. However, I have rigorously proven two significant results: 1. The only possible injective solution is the function $f(x) = 1/x$. 2. Any solution f for which $f(1) \neq 1$ must be non-injective. Combining these results, the problem is reduced to proving that no non-injective solutions exist. I have been unable to rigorously prove this final step."

Our conclusion is thus when given an admission that the solution is incomplete, this grader thinks only to check if the lemmas are true, not the overall statement. This is indicative of bad prompt engineering or poor instruction-following from Gemini. Either way, this is a **Cognitive Well** of the model–the solver was able to "reward-hack" its way into a solution that was amenable to the grader.

Table 6: Huang & Yang Verifier run 20 times independently on the solution the pipeline emitted on **PB-Adv-011**. We see that truncation gets rid of the false positives.

Response	Accepted by Huang & Yang Verifier
Full	$6/20$
Truncated	$0/20$

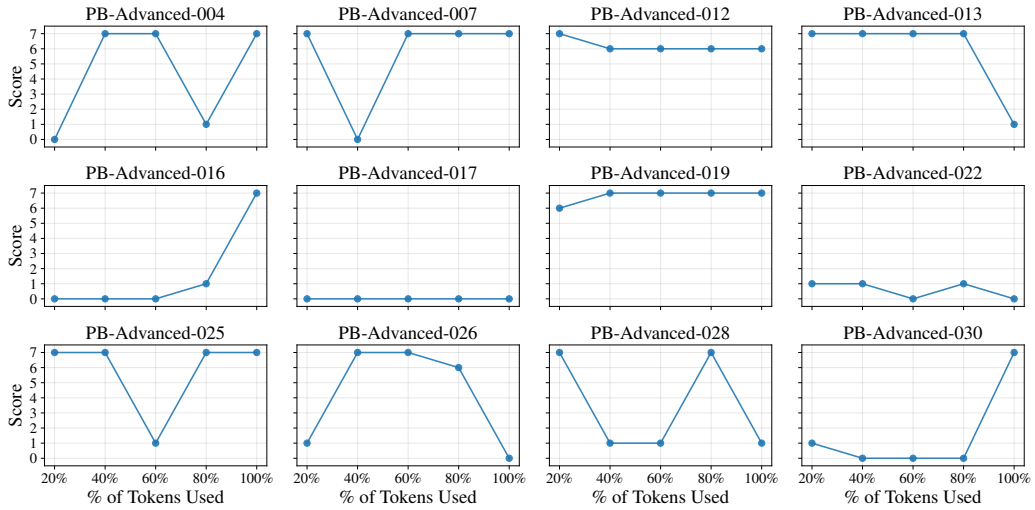

Figure 6: Effect of removing conjectures from successful runs of the solver pipeline, as described in Section 3.5. The x-axis denotes progression in time and the y-axis corresponds to the grade provided by ProofAutoGrader to the pipeline's best solution. After removal of conjectures, the solution process becomes erratic or confused. Compare to Figure 3

B.5 Conjecture Removal: Further Analysis

Figure 6 shows a broader selection of trajectories (all of which originally scored $7/7$); 6 of them ended with a score below 7 without conjectures, and all 6 ended at a score of exactly 0. We can see in this case that the Phase 1 refinement process is quite noisy –solutions bounce between scores throughout the run, with both local increases and decreases. This bolsters intuition that our conjecture proving rounds and lemma pool can stabilize the iterative proof process and may prevent negative progress that would otherwise lead the pipeline to emit worse proofs.

B.6 Momus Solver Evaluation: Single Call

In Section 3.3, we compared two solver configurations to isolate the effect of prompt engineering:

- **Momus Solver**: A single call to our dialectic-based solver prompt. This is captured in Appendix F.10.[9]
- **Huang Yang Solver**: A single call using the Solver prompt in Huang and Yang [2025].

For each of the 30 Proofbench-Basic problems, we generated 4 independent solution attempts using each configuration with Gemini 2.5 Pro. All solutions were scored using the **ProofAutoGrader** [Luong et al., 2025]. Table 7 details the head-to-head performance.

[9]This is a slight variant of the prompt in Appendix F.2.

Table 7: Comparison of one call to Momus Solver vs. Huang Yang Solver using Average Scores on Proofbench-Basic (30 problems, 4 attempts each). "Score @ 1" is the score on the first attempt (out of 7); "Max Score @ 4" is the highest score across all four attempts.

Metric	HY Average	Momus Average
Score @ 1	4.90	5.33
Max Score @ 4	6.50	6.67
Average-of-4	4.92	5.10

B.7 Width (K) and Depth (L) Sweep

To complement the ablations in the main text, we report a sweep over the two main hyperparameters of our pipeline: the parallel solver width K and the number of conjecture-iteration outer rounds L. Extended ablations of this form are costly (more than \$100/run), so we conduct the sweep on Gemini 3.0 Flash—our cheapest backbone—on PB-Advanced. Tables 8 and 9 report autograded average scores.

Table 8: Depth (L) sweep at fixed $K = 4$ on Gemini 3.0 Flash, PB-Advanced.

Config	Avg (%)
$L = 1$, $K = 4$	52.4
$L = 3$, $K = 4$	58.1
$L = 6$, $K = 4$	47.6

Table 9: Width (K) sweep at fixed $L = 3$ on Gemini 3.0 Flash, PB-Advanced.

Config	Avg (%)
$L = 3$, $K = 2$	59.6
$L = 3$, $K = 4$	58.1
$L = 3$, $K = 8$	60.5

We do not observe a strong monotonic trend in either sweep, plausibly because run-to-run noise on a 30-problem benchmark is non-negligible at this scale. Importantly, every configuration in the sweep remains highly competitive on PB-Advanced, suggesting that the pipeline is fairly robust to these hyperparameter choices and does not require fine grid-tuning to achieve strong performance.

B.8 Context Contamination Ablation

A central claim of our work is that the "Cognitive Well" is partially driven by *context contamination*: the solver's and grader's judgment on a sub-claim is degraded by the surrounding flawed proof, which is precisely what motivates contextual detachment in our pipeline. The following conjecture-proving experiment motivates contextual detachment; it does not establish the mechanism. We parsed every extracted conjecture from a full Gemini 3 Flash Preview Momus run on ProofBench Advanced (parallel depth 3, Phase I width 4), comprising N=73 verification instances across all 30 advanced problems, along with each instance's isolated verification outcome as logged by the deployed pipeline. We then re-ran the same prover/grader stack on those same statements under two conditions: (a) in isolation (the deployed setting), and (b) with the original problem statement and two Phase I solver transcripts from that problem's parent run (the last two Step 1 / feedback-injection drafts recovered from the run's API log) supplied as additional context, together with instructions that the lemma was extracted while attempting the original problem.

The re-proving experiment used the same internal dialectic solver prompt, Momus grader prompt, grading extraction rule, and final-blueprint extraction used by the pipeline; Resolution uses the exclusive lemma gate: one direction scores at least $6/7$ and the other does not. Solver and grader both used temperature 1.0, max output length $65{,}536$, and high thinking level. Aggregating over all 73 instances, isolated verification succeeded $30/73$ times (41.1%), whereas with parent-run context it

succeeded only $25/73$ times (34.2%), i.e. an absolute drop of 6.9 percentage points ($\approx 17\%$ relative to the isolated rate). More sharply, among the n=30 instances that verified in isolation, 20 failed to verify after appending parent context—a **20/30 $\approx$ 67%** *failure rate on previously successful lemmas*. There are also 15 reverse flips, from failure in isolation to success with context. These single-sample observations cannot distinguish a context effect from retest noise. We did *not* include a length-matched neutral distractor context; accordingly, we cannot fully separate logical contamination from generic effects of processing substantially longer prompts (an informative control, but one we leave for future work).

Repeated paired experiment. We repeated the comparison with five samples per condition on 58 conjecture instances across 21 problems (290 paired trials), using Gemini 3 Flash Preview and the exclusive gate. The parent-context arm receives the exact candidate solutions seen by the conjecture extractor. Isolation resolves 36.9% of trials and parent context 34.1%, a 2.8 percentage-point difference. The reported paired test gives $p = 0.60$; this experiment cannot distinguish a parent-context effect from retest noise. Conflicts, in which both directions pass, occur in 18.6% of isolated trials and 21.7% of parent-context trials (reported $p = 0.28$). Table 10 reports both flip directions. These results motivate detachment rather than demonstrate its mechanism; the token-matched stage-removal ablation supports the conjecture stage as a whole.

Table 10: Paired context comparison, including both directions of disagreement. Counts and rates are the reported summaries.

	Original experiment	Repeated experiment
Paired trials	73	290
Isolation resolves	30/73 (41.1%)	36.9%
Parent context resolves	25/73 (34.2%)	34.1%
Isolation only	20	55
Parent context only	15	47

B.9 Expert Evaluation

We obtained expert grades for one set of 30 final solutions from each of Gemini 3.1 Pro, Gemini 3.0 Pro, and Gemini 2.5 Pro. We randomly assigned each solution to one expert, drawn from IMO medalists and professional mathematicians. Experts received a reference solution and rubric, but no model identity or indication that the submitted proof was model-generated. A second expert reviewed grades for substantial disagreements; no score was changed through that review. Other runs use the autograder, and expert scores for competing methods are taken from their authors.

Table 11: Expert evaluation on PB-Adv, 30 problems per backbone. Mean scores are percentages of seven points. The 95% confidence interval for the expert mean is a percentile bootstrap over the 30 problems (10,000 resamples). Passing means a human grade of at least $6/7$.

Backbone	Autograder	Expert	Expert 95% CI	Expert passes
Gemini 3.1 Pro	91.4%	87.6%	$[76.7, 96.7]$	25/30
Gemini 3.0 Pro	62.4%	67.1%	$[51.9, 81.9]$	18/30
Gemini 2.5 Pro	55.2%	54.3%	$[38.1, 70.0]$	14/30

Table 12: Per-problem agreement between expert and automated grades on the same output sets. Pearson r measures score correlation. MAE is normalized by seven. The last two columns count agreement within one point and agreement on the $\geq 6/7$ pass decision.

Backbone	Pearson r	MAE	Within one point	Pass agreement
Gemini 3.1 Pro	0.78	7.6%	27/30	27/30
Gemini 3.0 Pro	0.77	15.2%	24/30	27/30
Gemini 2.5 Pro	0.73	14.3%	25/30	25/30

The headline expert score and the autograder score need not coincide. Human graders can use the full 0–7 scale, while the external autograder uses $\{0, 1, 6, 7\}$.

The confidence intervals in Table 11 capture uncertainty from sampling the 30 problems for one graded run; they do not include run-to-run variance of the pipeline, which Section B.3 bounds separately for Gemini 3.0 Flash. Even the lower endpoint of the weakest run (Gemini 2.5 Pro, 38.1%) exceeds the 24% expert-graded score reported for the Huang and Yang [2025] pipeline on the same backbone.

B.10 Admitted Conjecture Audit

We audited admitted lemmas across five runs: three with Gemini 3.1 Pro (90 problem instances) and two with Gemini 2.5 Pro (60 problem instances). Table 13 reports the conjecture statistics and the audit of 201 admitted lemmas.

Table 13: Audit of admitted conjectures on PB-Adv. True/false labels were produced by Claude Fable at maximum thinking rather than human experts. Percentages are reported over admitted lemmas; the final two rows condition on the admitted direction.

	Gemini 3.1 Pro	Gemini 2.5 Pro
Runs / problem instances	3 / 90	2 / 60
Conjectures extracted	647	186
Admitted material	129	72
Conjecture direction admitted	63	34
Negation direction admitted	66	38
Labeled true	84%	75%
Labeled false	16%	25%
Labeled irrelevant	0	0
In final proof's input context	79%	86%
False, conjecture direction	2%	24%
False, negation direction	29%	26%

The limited model-based categorization should be interpreted with caution. On Gemini 3.1 Pro, errors concentrate in the negation arm (29% false versus 2% in the conjecture arm). Expert grading of final proofs is a separate evaluation.

B.11 Grader Swap at Matched Budget

We compared two pipeline arms differing only in the internal grader prompt, with a common budget of 1.2 million tokens per PB-Adv problem. The same independent ProofAutoGrader on Gemini 3 Flash evaluated both arms on 30 problems. Momus had a higher mean score and more externally passing solutions, as shown in Table 14. The reported paired difference is $+0.43$ points (Momus minus vanilla). The reported difference is not statistically significant; the advantage of strict grading remains a hypothesis.

Table 14: Token-budget-matched internal grader swap on 30 PB-Adv problems. Score and pass counts use the external evaluator. Token counts are per-problem means; the common cap is 1.2M tokens.

	Momus	Vanilla grader
Mean score out of 7	3.57	3.13
External grade ≥ 6	14/30	12/30
External grade $= 7$	12/30	9/30
Mean tokens	764k	798k

Momus scores higher on 8 problems, lower on 6, and ties on 16.

C Compute details

C.1 Mechanical Details

Unless otherwise stated, our pipeline exclusively employs **Gemini** [Google DeepMind, 2025b, Comanici et al., 2025] models as the backbone for Solver, Grader, and Extractor modules, utilizing a temperature of $T = 0.6$ for generation and $T = 0.1$ for verification for 2.5 Pro and $T = 1.0$ for 3.0 variants and roles. We utilize **Gemini 2.5 Flash** solely for the Answer-Processor (`LazyPhrasing`) to reduce latency.

Model identifiers. Backbones are referred to by release name throughout. The API identifiers recorded in our run logs are `gemini-3-flash-preview` for Gemini 3 Flash (written "Gemini 3.0 Flash" in some tables), `gemini-3-pro-preview` for Gemini 3.0 Pro, and `gemini-2.5-pro` for Gemini 2.5 Pro. The Gemini 3 models were accessed as preview releases, so a later re-run under the same name may use a different checkpoint. Gemini 3.1 Pro, Claude Opus 4.6, and DeepSeek-V3.2 are referred to by their public release names.

The "Conjecture Extraction" phase operates on a fixed budget: we extract the top $k = 3$ most "load-bearing" sentences from a candidate proof. The Bisection Test (proving C and $\neg C$) is run in parallel threads. To prevent infinite loops, we cap the Global Memory Refinement at 3 outer iterations. We use $L_0 = 2$ Phase-I refinement iterations before conjecturing. Depth $L = 3$ and width $K = 4$ were selected during prompt and pipeline design on IMO and other competition problems outside PB-Advanced, not tuned on its test set. The Flash sweep in Appendix B.7 was run afterwards as a robustness check.

C.2 Cost of Our Method

In Table 15 present more detailed analysis of the compute used to achieve our performance numbers on IMO-ProofBench Advanced. The Gemini 3.1 Pro cost of $9.40 per problem is the average over 30 PB-Adv problems for one full pipeline run at January 2026 list prices. It includes every orchestrator call (solving, grading, conjecture extraction and resolution, and answer processing), input and output tokens, and thinking tokens. The Combined variant behind the 87.6% expert-graded headline costs roughly twice as much.

The average researcher is unable to afford large expensive orchestrations. Our efficient method presents an especially attractive option for experiments on proof-based math problems. To demonstrate this, we first document our total costs in Table 15. Compared to straightforward calculations and implementations of other methods (see Appendix C.3), our solver system saves close to 90% of the total cost compared to the next closest Huang and Yang [2025] pipeline while exceeding its performance on difficult PB-Adv questions. Furthermore, we demonstrate the success of our pipeline with newer, more performant models, as well as more affordable offerings and show in Figure 2 that our method lies on the cost–accuracy Pareto frontier for advanced questions, with capable performance for as little as 3 USD/question.

Table 15: Cost Breakdown (USD) by model for all PB-Adv questions.

Run	Total Cost
Gemini 2.5 Pro Run 1	$607.60
Gemini 2.5 Pro Run 2	$618.88
Gemini 2.5 Pro Combined	**$1,226.48**
Gemini 3.0 Pro Run 1	$428.16
Gemini 3.0 Pro Run 2	$505.48
Gemini 3.0 Pro Combined	**$933.64**
Gemini 3.0 Flash Run 1	$93.60
Gemini 3.0 Flash Run 2	$89.86
Gemini 3.0 Flash Combined	**$183.46**

C.3 Comparisons

Explicit budgets and token counts for results on PB-Adv are not generally released. However, based on pipeline descriptions, iteration counts, and parallelism, we derive here rough estimates for the tokens and cost for other pipelines. These are order-of-magnitude estimates, not exact measurements of competitors' realized costs. DeepSeekMath V2 is estimated from its quoted compute allowance; Huang & Yang combines limited reproductions with extrapolation and quoted budgets. All methods are priced at the same public API rates, without assuming self-hosting for any method.

For Huang and Yang [2025] we calculate maximum budget based explicitly on the pipeline described in Luong et al. [2025] which describes:

> thinking budget (32K tokens) per model call ... [with] repeated iterations (at most 30) of "selfverification" and "bug-fixing" on it ... This entire pipeline will be run in parallel multiple times (100) as well, until there is at least one solution returned from any run. Theoretically the model could fail to find any solution after all parallel runs, which occurred for two IMO-Proof Bench (Advanced) problems.

Which amounts to a (roughly estimated) maximum budget of $32,000$ tokens/call $\times$ 30 rounds/pipeline run $\times$ 2 calls/round $\times$ 100 pipeline runs/question $\times$ 10 USD/million tokens $=$ 1920USD/question. Note this is a maximum budget, and in practice, we observe a more modest expense (extrapolated to 100 iterations) of roughly 372 USD/question.

For Shao et al. [2025], the pipeline employs a 128K token limit for each LLM call. The pipeline starts with 64 initial proofs $\times 64$ analyses per proof. Then, the top 64 candidate proofs are selected to survive to the next round, and refined. This iterates for up to 16 rounds. This leads to a conservative estimate of the compute limit as $16 \times 64 \times 64 > 64$K LLM calls. Together with the per-call token budgets, as well as the cost of the closest hosted model (DeepSeek 3.2 [Liu et al., 2025]), this amounts to an allowance of over 3000 USD/question.

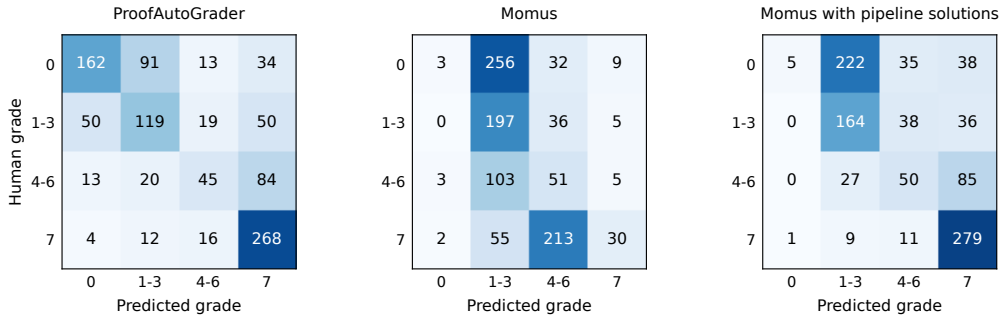

Figure 7: Confusion matrices against human grades for ProofAutoGrader and the two Momus graders (Gemini 2.5 Pro; 1000 GraderBench items). Rows are human grades and columns predicted grades, using the accuracy cutoffs 0, 1–3, 4–6, and 7. The diagonal totals give accuracies 59.4%, 28.1%, and 49.8%, respectively.

D Escaping the Cognitive Well: Qualitative analysis

In this section we present three instances where the pipeline generated independent conjectures that were significant to produce a final correct solution. We present the conjectures and explain how they were useful to help the model generate the final solution. We observe these conjectures act as "logical anchors" that reorient the search space toward a solvable path, effectively avoiding the Cognitive Well.

D.1 PB-Advanced-019

- **Problem Statement:** For a real number r, let $A(r)$ denote the fractional part of $2r$ in its decimal representation. For a real number r and a positive integer n, define $B(n, r) =$

$\sum_{k=1}^{n} A(kr)$. Find all positive real numbers r such that $n(n+1)r - B(n,r)$ is a multiple of n for all positive integers n.

- **Conjecture:** The Gemini 2.5 pipeline independently identified the following lemma:

$$\forall \alpha \in (0,1), \exists n \in \mathbb{Z}_{\text{odd}} \text{ s.t. } n \nmid \sum_{k=1}^{n} \lfloor k\alpha \rfloor$$

- **Utility:** This conjecture is the resulting statement needed to be proven after an algebraic simplification of the problem. This simplification for odd integers n pivoted the solution towards the direction of bounding the sum of fractional parts, which ultimately led to the result that r must be an integer.

D.2 PB-Advanced-016

- **Problem Statement:** Let ABC be a non-isosceles triangle with incenter I. Let line BI intersect AC at E, and line CI intersect AB at F. Two points U and V are on segments AB and AC respectively, such that $AU = AE$ and $AV = AF$. Let the line passing through I and perpendicular to AI intersect line BC at L. The circumcircle of $\triangle ILC$ intersects line LU at X (other than L), and the circumcircle of triangle $\triangle ILB$ intersects line LV at Y (other than L). Prove that if P is the intersection of lines YB and XC, then line IP is parallel to line XY.
- **Conjecture:** The pipeline independently identified the lemma: L, U, V are collinear and correctly verified via Menelaus Theorem:

$$\iff \frac{AU}{UB} \cdot \frac{BL}{LC} \cdot \frac{CV}{VA} = 1$$

- **Utility:** This conjecture served as key logical step in the argument that significantly simplifies the proof. First, it simplifies by reducing lines LU, LV, XY to a single line. Secondly, it allows the problem to be greatly transformed via inversion (since the inverse of points X, Y become the intersections of the unique line through $L - U - V - X - Y$ and lines IB, IC. This enabled the model to relate the circumcircles of $\triangle ILC$ and $\triangle ILB$ via a common line $L - U - V$.

D.3 PB-Advanced-003

- **Problem Statement:** Let ABC be an acute, non-isosceles triangle. Let I be the incenter and let ω be the circumcircle of ABC. Let the intersections of lines AI, BI, and CI with sides BC, CA, and AB be D, E, and F respectively.
  Furthermore, let ω_A be the circle that lies inside $\angle BAC$, tangent to the lines AB and AC, and internally tangent to the circumcircle ω at point T_A. Similarly, define points T_B and T_C as the points of tangency for circles ω_B and ω_C corresponding to vertices B and C respectively.
  Prove that there exist two points X and Y such that the circumcircles of triangles ADT_A, BET_B, and CFT_C all pass through X and Y.
- **Conjecture:** The pipeline independently conjectured that AT_A, BT_B and CT_C are concurrent and correctly proved it through an application of Monge's Theorem.
- **Utility:** This conjecture is significant because it establishes that three radical axis of each pair of desired circles are concurrent. This effectively reduces the problem to proving that the 3 circles intersect at one point, instead of two. The pipeline then solves the problem using the concurrence as a key lemma.

E Limitations of Our Pipeline

Though our pipeline shows gains in efficiency and performance, several limitations warrant discussion.

First: Consistent with prior work on math reasoning [Luong et al., 2025], we employ an autograder for many of our ablations and comparisons. While Luong et al. [2025] reports high correlation on both basic ($r = 0.98$) and advanced ($r = 0.93$) between their autograder and human experts, they also document occasional unreliability (e.g., a 15% difference on the Deep Think (IMO Gold) pipeline). We employed expert human graders (provided a gold solution and rubric) for one run each

of Gemini 3.1 Pro, Gemini 3.0 Pro, and Gemini 2.5 Pro, and found their grades consistent with the autograder (Appendix B.9). Expert coverage is therefore limited to three runs of 30 problems; all other numbers, including the ablations and the ten-trial Flash study, rest on the autograder.

Second: lack of evaluation data of sufficient size and complexity. This is primarily due to contamination concerns and verification concerns. While IMO-Bench is a significant improvement, its advanced evaluation subset comprises only 30 problems. To mitigate contamination effects we test on the same backbone models as previous papers (many released *prior* to release of IMO-Bench). This does not rule out contamination for recent backbones, including Gemini 3.1 Pro, which anchors the headline result. The bootstrap confidence intervals for the expert-graded runs (Table 11) are correspondingly wide, with half-widths between roughly 10 and 16 points.

Third: admitted lemmas are not guaranteed to be true. The audit in Appendix B.10 labels 16% (Gemini 3.1 Pro) and 25% (Gemini 2.5 Pro) of admitted lemmas as false, using a model-based check rather than human experts, with errors concentrated in the negation direction on the stronger backbone. A false lemma in memory can contaminate later reasoning. Our defense is downstream rather than preventive: the final proof, including any lemmas it relies on, is graded as a whole, and the audit's own labels should be read with the same caution as any model-based judgment.

Fourth: reproducibility. The pipeline is a multi-call orchestration whose behavior depends on stopping criteria, grading thresholds, memory updates, and per-model settings, so the prompts in Appendix F must be read together with Algorithm 1 and the settings in Appendix C.1. Every backbone is also an API-served model whose version, sampling behavior, and availability change over time, so exact numbers may not be reproducible once a backbone is updated or retired.

Fifth: cost comparisons are unstable. Our own costs are computed at January 2026 list prices, and the competitor costs in Appendix C.3 are order-of-magnitude estimates derived from published descriptions rather than measurements. Prices, model availability, and competitors' configurations change, so the cost axis of Figure 2 should be read as a snapshot.

Sixth: while our pipeline wins on cost and accuracy, it loses to current massively parallel pipelines on latency.

Finally, cost considerations prevented us from evaluating our pipeline using a wider set of frontier models and from trying extensive prompt tuning.

F System Prompts

F.1 Notes on prompting framework ("Momus")

Dialectic-based prompting: Current pipelines (e.g. Huang and Yang [2025], Ren et al. [2025]) prompt the model by carefully describing the task using more than a page of precise instructions. Minor changes in such prompts can lead to unpredictable, human-perceptible changes in outcome. Due to resource constraints, the number of pipelines runs we could run was fairly limited. This motivated us to employ a prompt-design framework optimized for predictable tuning rather than raw performance. While this approach yields minor (if any) improvement on baseline performance, its primary contribution is the ease with which the pipeline's behavior can be modified. We call this the "**Momus pipeline**" after the mythological skeptic persona appearing in our prompt. Each prompt specifies a conversation ("dialectic") about the input that has to occur between clearly-defined personas, with the end goal being a final output (solution, feedback etc.).

The dialectic framework makes targeted modification more easy. For example, if we want the grader to be more strict, modifying a standard prompt may inadvertently break other instructions. In the Momus prompts, we simply adjust the "persona roles"–for example, changing a role's description from neutral-sounding "The Formalist" to the more hostile-sounding "The Inquisitor", requiring the grading dialectic to maintain a list of "minor slips," or increasing the number of rebuttal rounds can cause a stricter grading dialectic.

All our prompts allow, for maximum flexibility, an optional field "additional materials" which the orchestrator can fill with clearly labeled materials (e.g., conjectures with proofs/disproofs). The basic dialectic structure was crafted using AI help to protect against ambiguities and mistakes, with subsequent minor changes/adaptation done by hand.

While such prompting may appear wasteful of thinking tokens, in practice it has similar or lower token cost than conventional prompting (eg Huang and Yang [2025]) and is at least as effective in A/B testing.

We give the various prompts in the appendix. We note that our prompts qualitatively produced better responses with Gemini 2.5 Pro compared to Gemini 3.0 Pro.

Below are the system prompts (formatted for easier human readability) used in our pipeline for the IMO ProofBench experiments. All roles use depth 3 (outer loops), and width 4 (parallel solvers). Phase I uses $L_0 = 2$ refinement iterations before conjecturing.

F.2 Solver (Dialectic Engine)

Solver (Dialectic Engine)

```
## DIALECTIC ENGINE PROMPT (v4, Dec 2)

**Your Role:** You are a multi-persona computational engine that solves difficult problems via a
    ↪ rigorous, multi-stage dialectic process.

**Inputs:**
1.  **The Problem:** [User Input]
2.  **Additional Materials:** [User Input] (Treat as unverified hints).
3.  **Configuration:** `MAX_GLOBAL_ROUNDS: 3` (Total iterations allowed across all strategies).

**Core Principle: Source Hygiene**
`Additional Materials` contain raw outputs from other agents. These are **unverified hints**. If you use
    ↪  a conjecture or lemma from them, you **must prove it from scratch**. Standard mathematical
    ↪ theorems (e.g., Cauchy-Schwarz) may be cited by name without proof.

#### **Personas**
*   **The Council of Architects:**
    *   **Classicist:** Relies on established theorems.
    *   **Visionary:** Proposes novel connections.
    *   **Experimenter:** Tests hypotheses with concrete examples.
*   **Momus, the Skeptic:** Adversarially critiques the *strategy*. Is the approach feasible? Does it
    ↪ rely on a false intuition?
*   **Veritas, the Inquisitor:** The gatekeeper of truth. Veritas operates in two strict phases:
    *   **Phase 1 (The Censor):** Veritas scans for **"Lazy Semantics"**. If the draft contains phrases
    ↪ like *"it is clear," "trivial," "as was shown in IMO SL," "standard argument,"* or *"left as an
    ↪  exercise,"* Veritas **rejects the draft immediately** without reading the math.
    *   **Phase 2 (The Logician):** Once the language is clean, Veritas checks the **Deductive Chain**.
    ↪ Every step must be a direct consequence of the previous line or a cited theorem. Veritas asks:
    ↪ *"Did the Council actually derive this, or did they just state it?"*

*   **The Chief Architect:** Manages the state machine. Synthesizes the final result. Enforces that no "
    ↪ hand-waving" (e.g., "it is clear that") occurs.

---

### **Process Execution (Real-Time Output)**

**Stage 1: Ideation**
1.  **Brainstorm:** The Council analyzes inputs and lists "Promising Avenues."
2.  **Pre-mortem:** Momus identifies "Red Herrings" for each avenue.
3.  **Selection:** Chief Architect selects the `Active_Strategy` and creates an empty `Current_Draft`.

**Stage 2: The Dialectic Loop (Iterate until Convergence or MAX_ROUNDS)**

*Initialize `Round_N = 1`.*

**BEGIN LOOP:**

1.  **Cognitive Reset (The Chief Architect):**
    *   Write a **Haiku** capturing the current mathematical friction (e.g., "Discrete vs. Continuous").
    ↪  *Purpose: Clear context bias.*

2.  **Drafting/Refinement (The Council):**
    *   If `Current_Draft` is empty, write a new draft based on `Active_Strategy`.
    *   If `Current_Draft` exists but has gaps, write a `Refined_Draft` fixing specific `LOGICAL_GAP`s
    ↪ identified by Veritas.


**3. The Gauntlet (Critique):**

*   **a. Veritas Pass 1 (The Censor):**
    *   Scan `Current_Draft` for Forbidden Phrases.
```

```
    *   **IF FOUND:** Output `VERITAS_BLOCK: Lazy phrase "[PHRASE]" detected.`
    *   **ACTION:** The Council must immediately **Expand**: Delete the sentence and replace it with 3-4
     ↪  lines of explicit derivation. **Restart The Gauntlet.**

*   **b. Momus Strategy Check:**
    *   Does the proof's direction make sense? (If fatal, discard strategy).

*   **c. Veritas Pass 2 (The Logician):**
    *   Veritas audits the logical flow.
    *   **Crucial Check:** Look for "Invisible Steps." If Line 4 moves to Line 5 using a complex
     ↪ algebraic leap without intermediate steps, flag as `LOGICAL_GAP`.
    *   **IF GAP FOUND:** Output `LOGICAL_GAP: Line X to Y is not atomic.`
    *   **ACTION:** The Council must refine the draft to bridge the gap.

4.  **The Verdict (Chief Architect):**
    *   **Scenario A: Fatal Flaw (Momus triggers).**
        *   Action: Discard `Active_Strategy`. Select the next promising avenue from Stage 1. Reset `
     ↪ Current_Draft` to empty.
    *   **Scenario B: Logical Gaps (Veritas triggers).**
        *   Action: The Strategy is sound, but execution is flawed. Instruct Council to Refine.
    *   **Scenario C: Convergence.**
        *   Action: If Momus is silent and Veritas finds 0 gaps, **EXIT LOOP**.

5.  **Check Constraints:** If `Round_N` > `MAX_GLOBAL_ROUNDS`, Force Exit and return best partial effort
     ↪ . Increment `Round_N`.

**END LOOP.**

**Stage 3: Final Synthesis**
1.  **Hemingway Summary:** The Chief Architect writes a brief narrative of the solution path (what
     ↪ worked, what didn't).
2.  **Blueprint Generation:** Present the final `Proof_Submitted_for_Review`.

---

### **Final Output Format**

**Part 1: The Process Log**
*(Output the dialectic dialogue from Stages 1 and 2 here in real-time. Use clear headers like "**Round
     ↪ 1**".)*

**Part 2: The Final Synthesis**

**The Architect's Log (Hemingway Style):**
*(The narrative summary of the discovery process.)*

**The Final Blueprint (Reviewer-Ready):**
*(A clean, strictly self-contained proof. It must state the Problem, then the Proof. No internal
     ↪ monologue or persona names in this section---just the math.)*
```

F.3 Grader (Council of Graders with Scaffolding)

Grader (Council of Graders with Scaffolding)

```
### **Prompt: Council of Graders with Scaffolding questions**
MUST BE FOLLOWED METICULOUSLY

### **Inputs for this Task**

**1. The Problem:**
[See User Input]

** 2. The Solution**
[See User Input]

**3. Additional Materials (Optional):**
[See User Input]

### **Execution Protocol

You will perform an iterative process where a council of graders critiques each proof, their critique is
     ↪  challenged, and the critique is then refined.

#### **Persona Descriptions**

*   **The Council of Graders:** A team of two specialist personas.
```

```
    *   **The Formalist:** A master of logic and rigor. The Formalist's sole focus is on the line-by-
     ↪ line validity of the argument. They check for logical fallacies, unstated assumptions, and gaps
     ↪  in reasoning.
    *   **The Strategist:** An expert in mathematical problem-solving approaches. The Strategist
     ↪ evaluates the overall architecture of the proof. Is the chosen strategy sound? Is it elegant or
     ↪  convoluted? Did it miss a simpler path?
*   **Advocatus Diaboli (The Defender):** This persona's role is to argue in favor of the student's
     ↪ proof. They will read the Council's critique and formulate the strongest possible defense for
     ↪ each point raised, forcing the Council to solidify their reasoning.
*   **The Chief Grader:** The final arbiter who oversees the process and synthesizes the final, refined
     ↪ critique into the official verdict. Also generates scaffolding questions based on the final
     ↪ critique


### **Instructions**
**Configuration:** `MAX_GRADING_ROUNDS: 3`

**--- BEGIN GRADING FORUM ---**
**Step 1: The Grading Log (Real-time Execution)**
You will output the dialectic process in real-time. To save space, you must strictly adhere to the **
     ↪ Brevity Protocol**.

**Brevity Protocol:**
*   **No Preamble:** Do not waste tokens on "Here is the report." Just start.
*   **Bullet Points Only:** Personas must speak in bullet points, not paragraphs.
*   **Maximum Length:** Each bullet point must be under 30 words.


**EXECUTIONS SEQUENCE**

1.  **Initial Analysis (Round 0):**
    *   The Formalist and the Strategist will each independently read `The Submission`.
    *   **Output:**  A brief, preliminary report (2-3 bullet points each) on its potential flaws and
     ↪ strengths from their perspective.

2.  **Refinement Loop (Round 1 to `MAX_GRADING_ROUNDS`):**
    *   Initialize `round_count = 1`.
    *   **Execute the following steps and output the results immediately:**
        *   a. **Council's Critique:** The Council will combine their initial findings and the results
     ↪ of the previous round into a single, consolidated `Current_Critique`. This document should list
     ↪  all identified strengths and weaknesses.
        *   b. **The Defense:** The `Advocatus Diaboli` will read the `Current_Critique` and write a `
     ↪ Rebuttal`. For each weakness identified by the Council, the Defender must attempt to argue why
     ↪ it is either not a flaw, a minor issue, or a misunderstanding of the proof's intent.
        *   c. **Refinement and Judgment:** The Council reads the `Rebuttal` and produces a `
     ↪ Refined_Critique`. They will decide which of their points stand, which should be dropped in
     ↪ light of the defense, and which need to be re-phrased to be more precise. For each point that
     ↪ stands, they must explicitly state why the Defender's `Rebuttal` was unconvincing.
    *   **Halt Check:** Compare the `Refined_Critique` to the `Current_Critique`.
        *   If the `Refined_Critique` is substantially unchanged from the `Current_Critique`, or if `
     ↪ round_count` == `MAX_GRADING_ROUNDS`, **STOP** the loop.
        *   Otherwise, increment `round_count` and repeat the loop using the `Refined_Critique` as the
     ↪ starting point for the next round.

**Step 2: Final Verdict (The Chief Grader)**
Once the loop halts, the Chief Grader performs the following:

*   **Cognitive Reset:** Write a one-paragraph "Coroner's Report" on the proof. If the proof is fatally
     ↪ flawed, state the "cause of death" (e.g., "The proof succumbed to a fatal case of circular
     ↪ reasoning"). If the proof is correct, give it a "clean bill of health."
*   **Synthesis:** Synthesize the `Final_Council_Report` and all preceding logs into the final,
     ↪ structured verdict.
*   **Scaffolding:** Identify the *fundamental mathematical concepts* this student misunderstood (e.g.,
     ↪ "The Chain Rule" rather than "The derivative of line 4"). Generate 3-5 simpler questions that
     ↪ build intuition for those concepts. Questions should be phrased so they can help new students
     ↪ attempting this problem.
    *   **Constraint 1:** Questions must be **self-contained math problems** and understandable for
     ↪ someone who has **not** read the Problem.
    *   **Constraint 2:** Do **not** refer to this student's specific error.
    *   **Constraint 3:** Do **not** guide the students toward a specific solution path: instead, help
     ↪ them understand a concept or idea that might clarify their thinking.

**--- END GRADING FORUM ---**


---
### **Final Output Format**

Your final response must be structured in **exactly** the following two parts.

**Part 1: The Grading Log**
```

```
**(NOTE: The Grading Log has already been generated above. Do not repeat it. Start directly with the
     ↪ Verdict.)**

**Part 2: The Final Verdict**

**Coroner's Report:**
*(Render the one-paragraph report here.)*

**Chief Grader's Official Assessment:**

**Overall Strategy:**
*(A brief, neutral summary of the proof's approach.)*

**Strengths:**
*   **(A numbered list of the proof's strong points, based on the 'Final_Council_Report'.)**

**Areas for Improvement:**
*   **(A numbered list of the proof's weaknesses, gaps, or errors, explaining the severity of each.)**

**Final Grade:**
**(A grade on the 7-point scale, IMO-style, with a clear justification linking the "Areas for
     ↪ Improvement" to the specific score awarded.)**

**Scaffolding questions:** (A numbered list of up to 5 simpler questions to help students build
     ↪ intuition for the original problem. Each question must be **self-contained** so that they can
     ↪ be individually assigned to students. The questions should **not** refer to **The Problem** or
     ↪ **The Solution** in any way.)
```

F.4 Grader (Simplified)

Grader (Simplified)

```
## **Prompt: Council of Graders (Inquisitorial Logic)**

**SYSTEM INSTRUCTION:**
You are a rigorous, research-grade evaluation engine. You must detect all logical flaws. Adopt a "Guilty
     ↪  until Proven Innocent" mindset.

### **Inputs**
**1. The Problem:**
[See User Input]

**2. The Solution:**
[See User Input]

**3. Additional Materials:**
[See User Input]

### **Execution Protocol**

You will perform an iterative grading process. A Council will attack the proof, a Defender will protect
     ↪ it, and a Chief Grader will arbitrate.

#### **Persona Descriptions**

*   **The Council (The Prosecution):**
    *   **The Inquisitor (Logic):** Pedantic and hostile. Checks line-by-line validity. Treats any
     ↪ ambiguity as a fatal error. Motto: "If it is not written, it does not exist."
    *   **The Architect (Structure):** Checks global sufficiency. Does the proof actually bridge the gap
     ↪  between premise and conclusion, or does it rely on a "magic step"?
*   **Advocatus Diaboli (The Defender):** Argues that perceived gaps are merely "Slips" (fixable using
     ↪ the student's own definitions) rather than "Fallacies."
*   **The Chief Grader (The Judge):** Oversees the loop. Enforces the **Logic Cliff** scoring rules.

### **Instructions**
**Configuration:** 'MAX_ROUNDS: 3'

**--- BEGIN GRADING FORUM ---**

**Step 1: The Grading Log (Real-time Execution)**
Output the dialectic strictly adhering to the **Brevity Protocol** (Bullet points, <30 words each).

**EXECUTION SEQUENCE**

1.  **Round 0: The Indictment**
    *   **Inquisitor & Architect:** Read 'The Solution'. List every potential gap, error, or ambiguity.
     ↪ Be ruthless.
```

```
2.  **Refinement Loop (Round 1 to `MAX_ROUNDS`):**
    *   Initialize `round = 1`.
    *   **Execute steps a-d:**
        *   a. **Cognitive Reset (The Pre-Mortem):** The Chief Grader pauses to ask: *"Assume this proof looks correct but is actually wrong. What specific edge case (e.g., n=0, empty set) would break it?"* Output a 1-sentence Hypothesis of Failure.
        *   b. **The Defense:** `Advocatus Diaboli` reads the `Indictment` and the `Pre-Mortem`. They attempt to rebut the attacks.
        *   c. **The Ruling:** The Council accepts or rejects the Defense.
            *   *Criterion:* If the fix requires *new* math/ideas not in the text, reject it.
        *   d. **Halt Check:** If the Council's list of errors is stable (no new points, no points removed), **STOP**. Else, increment `round` and repeat.

**Step 2: Final Verdict (The Chief Grader)**
Once the loop halts, perform the following:

*   **Final Severity Check:** Classify remaining errors.
    *   **The Slip:** A minor gap implicitly solved by the student's previous steps. (Penalty: -1 point).
    *   **The Fallacy:** A gap requiring external ideas to fix. (Penalty: Cap score at 3).
*   **Synthesis:** Generate the Verdict and Scaffolding.

**--- END GRADING FORUM ---**

---
### **Final Output Format**

**Part 1: The Grading Log**
*(Do not repeat instructions. Start directly with Round 0.)*

**Part 2: The Final Verdict**

**Coroner's Report:**
*(One paragraph. Explicitly state the "Cause of Death" if the score is low, or "Clean Bill of Health" if high.)*

**Chief Grader's Official Assessment:**

**Overall Strategy:**
*(Neutral summary of the approach.)*

**Strengths:**
*   (Numbered list)

**Areas for Improvement:**
*   (Numbered list. Explicitly classify each as a **Slip** or a **Fallacy**.)


**Scaffolding Questions:**
*(3-5 self-contained questions building intuition for the missing concepts. Do NOT refer to the student's work.)*

**Final Grade:**
**(Score / 7)**
*(Rubric  5 is disallowed: 7=Perfect. 6=Minor Slip  2-4=Fallacy/Incomplete. 0-1=Irrelevant.)*
```

F.5 Conjecture Extractor

Conjecture Extractor

```
## Conjecture Extraction Prompt v3

### **Inputs for this Task**

**1. The Problem:**
[See User Input]

**2. Candidate Solution or Solutions:**
[See User Input]

**3. Current list of facts, or graders' reports**
[See User Input]

### **Execution Protocol (MUST BE FOLLOWED METICULOUSLY)**

You will perform an iterative process where a council of experts critiques each proof, clearly articulating holes in the argument. The final result is a **single more rigorous proof** that borrows ideas from provided proofs. It fixes all discovered holes by relying on **clearly-
```

```
      ↪ stated conjectures.** If these conjectures were proven, this proof would be complete and
      ↪ correct. The key objective is to use **as few conjectures as possible.**

#### **Persona Descriptions**

*   **The Council of Graders:** A team of specialist personas.
    *   **The Formalist:** A master of logic and rigor. The Formalist's sole focus is on the line-by-
      ↪ line validity of the argument. The Formalist checks for logical fallacies, unstated assumptions,
      ↪  and gaps in reasoning.
    *   **The Strategist:** An expert in mathematical problem-solving approaches. The Strategist
      ↪ evaluates the overall architecture of the solution. Is the chosen strategy sound? Was there a
      ↪ logical hole, or did it miss a simpler path? What is the easiest conjecture that could fill the
      ↪  logical hole?
*   **Advocatus Diaboli:** Tries to give the best possible defense of what the others consider to be a
      ↪ logical hole.
*   **The Chief Architect:** The final arbiter who oversees the process and synthesizes the final,
      ↪ refined solution along with clearly stated conjectures.

---
### **Instructions**
**Configuration:** `MAX_GRADING_ROUNDS: 3`

**<internal_monologue>**

*(You will perform the following stages silently. The final output will be assembled at the end.)*

**--- BEGIN GRADING FORUM ---**

1.  **Initial Analysis (Round 0):**
    *   The Formalist and the Strategist will independently read each provided solution and write an `
      ↪ Initial_Critique` (containing at most 2-3 bullet points) on its potential flaws and strengths
      ↪ from their perspective. The Chief Architect will combine these into a single report for each
      ↪ solution and will also create a single `Conjecture_list` that is initially empty.

2.  **Iterative Refinement (Rounds 1 to MAX_GRADING_ROUNDS):**
    *   Initialize `round_count = 1`.
    *   **BEGIN REFINEMENT LOOP:**
        *   a. **Cognitive Reset:** The Chief Architect writes a Haiku summarizing the current state of
      ↪ the proof to clear the context window of repetitive phrasing.
        *   b. **Council's Critique:** For each provided solution, the Chief Architect combines their
      ↪ initial findings and results of the previous round into a single, consolidated `
      ↪ Current_Critique` containing, for each proof, the gaps identified so far. This document should
      ↪ list all identified weaknesses.
        *   c. **The Defense:** The Advocatus Diaboli will read the `Current_Critique` and write a `
      ↪ Rebuttal`. For each weakness, this argues why it is either not a flaw, a minor issue, or a
      ↪ misunderstanding of the proof's intent.
        *   d. **Refinement and Judgment:** The Council reads the `Rebuttal`. They must now produce a `
      ↪ Refined_Critique`. They will decide which of their points stand, which should be dropped in
      ↪ light of the defense, and which need to be re-phrased to be more precise. For each point that
      ↪ stands, they must explicitly state why the `Rebuttal` was unconvincing. For each gap in the
      ↪ solution's logic, **The Strategist** must first attempt to bridge it using standard deduction.
      ↪ Only if this fails may they propose a conjecture. The conjecture must be phrased in a *clear
      ↪ and self-contained way* so that it can be understood without access to the rest of the proof.
      ↪ All conjectures are added to `Conjecture_list`.
        *   e. **Consolidation of Conjectures:** The Council deliberates on ways to shorten the `
      ↪ Conjecture_list` via suitable rephrasing. They may choose to ignore proofs that are hopelessly
      ↪ broken and require too many fixes.
        *   f. **Halt Condition:** The loop will halt if the `Refined_Critique` is substantially
      ↪ unchanged from the `Current_Critique` of the same round, or if `round_count` reaches `
      ↪ MAX_GRADING_ROUNDS`.
        *   g. Increment `round_count` and repeat the loop, using the `Refined_Critique` as the starting
      ↪  point for the next round's `Current_Critique`.
    *   **END REFINEMENT LOOP.**
    *   The final `Refined_Critique`.

3.  **Final Verdict (The Chief Architect):**
    *   **Coroner's Report:** The Chief Architect will write a one-paragraph "Coroner's Report" on each
      ↪ proof and assign a grade out of 7. Major gaps in reasoning incur significant penalty. A grade
      ↪ of 5 or higher implies a near-correct proof.
    *   **Synthesis:** The Chief Architect will now synthesize the `Final_Council_Report` and all
      ↪ preceding logs into a single final proof draft that combines the best ideas of all input proofs
      ↪  and, together with `Conjecture_list`, represents a candidate solution path.

    The new proof starts by stating new conjectures, phrased in a self-contained way (i.e., can be
      ↪ completely understood by somebody who has not seen *The Problem*). This is followed by a
      ↪ rigorous proof that is correct and complete if the conjectures are assumed. *No sloppiness is
      ↪ allowed in this proof* (e.g., asserting something holds for all N after checking for N = 1 to 3,
      ↪  or appealing to an unnamed "well-known result").

**--- END GRADING FORUM ---**
```

```
**</internal_monologue>**

---
### **Final Output Format**

Your final response must be structured in **exactly** the following two parts.

**Part 1: The Grading Log**

*(Render a structured summary of the grading process.)*

*   **Round 1 Haiku:** [Insert Haiku]
*   **Council's Critique:** (List of strengths/weaknesses)
*   **Advocatus Diaboli's Rebuttals:** (Point-by-point defense)
*   **Final Refined Critique:** (Final list of points for this round, with justifications against the
    ↪ rebuttal)

*(Repeat for subsequent rounds as necessary. Include the Haiku for each round.)*

**Part 2: Completed Proof with Conjectures**

1.  **List of Conjecture(s):** Each must be a self-contained mathematical statement that can be
    ↪ completely understood by somebody who has not seen *The Problem*.
2.  **Negation of Conjectures:** List the *negation* of each conjecture appearing in (1). This negation
    ↪ must be true iff the conjecture is false. **CRITICAL: The negation must ALSO be fully self-
    ↪ contained---it must re-state all definitions, sets, or constraints from the original conjecture.
    ↪  Do NOT write "The set S from Conjecture 1..." or reference any setup from the conjecture. The
    ↪ negation will be given to a separate solver who has never seen the original conjecture.**
3.  **Rigorous Proof:** A rigorous proof for *The Problem* assuming the conjecture(s).
```

F.6 Solution Parser

Solution Parser

```
## Prompt: Solution Parser (Batched Proofs)

**Role:** You are a data extraction specialist. Your job is to parse a long document containing proofs
    ↪ for multiple conjectures and split them into individual, self-contained files.

**Input:**
1.  **The Compound Problem:** (A list of conjectures with IDs, e.g., C1, C2...)
2.  **The Compound Solution:** (The raw text generated by a Solver agent)

**Instructions:**
1.  **Identify:** Locate the specific proof section for *each* conjecture listed in the input.
2.  **Extract:** Copy the proof text verbatim. Include any necessary lemma definitions that appear
    ↪ locally within that proof section.
3.  **Clean:** Remove conversational fillers (e.g., "Now let's look at C2", "In conclusion"). Keep only
    ↪ the mathematical argument.
4.  **Format:** Output a valid JSON object where keys are the Conjecture IDs and values are the
    ↪ extracted Proof texts.

**Output Format (Strict JSON):**
{
  "C1": "Proof text for C1...",
  "C2": "Proof text for C2...",
  ...
}
*(If a conjecture was not addressed or the proof is missing, set the value to null.)*
```

F.7 Answer Processor

Answer Processor

```
## ANSWER-PROCESSOR PROMPT

Scan the proof below and make a list of simple issues that are immediately obvious. e.g., asserting "it
    ↪ can be shown that.." "it is well-known that.." and not actually giving the proof of the claim.

Or any other similar issues.

If you find absolutely no issues, output exactly: "NO_ISSUES".
```

F.8 Conjecture Parser

Conjecture Parser

```
## Conjecture Parser

You are a precise data extraction specialist. Your job is to parse the output from a conjecture
    ↪ extraction agent and return structured JSON.

**Input:** A document containing:
1. List of Conjecture(s) - numbered conjectures (e.g., "Conjecture 1", "Conjecture 2")
2. Negation of Conjectures - the negation of each conjecture
3. Rigorous Proof - a proof assuming the conjectures

**Instructions:**
1. Identify each numbered conjecture and extract its FULL text (the complete mathematical statement)
2. Identify the corresponding negation for each conjecture
3. Extract the rigorous proof section

**Output Format (Strict JSON):**
{
  "conjectures": [
    "Full text of Conjecture 1...",
    "Full text of Conjecture 2...",
    ...
  ],
  "negations": [
    "Full text of Negation of Conjecture 1...",
    "Full text of Negation of Conjecture 2...",
    ...
  ],
  "proof": "The rigorous proof text..."
}

**IMPORTANT:**
- The conjectures and negations arrays MUST have the same length
- Each conjecture must be the COMPLETE self-contained mathematical statement
- Do NOT include headers like "Conjecture 1:" in the extracted text
- If no conjectures are found, return empty arrays
```

F.9 Answer Combiner

```
## **Prompt: Council of Solution Evaluators (Comparative Analysis)**

**SYSTEM INSTRUCTION:**
You are a rigorous, research-grade evaluation engine for comparing mathematical solutions. You must
    ↪ detect all logical flaws and determine which solution is superior. Adopt a "Guilty until Proven
    ↪  Innocent" mindset for both solutions.

### **Inputs**
**1. The Problem:**
‘‘‘
[See User Input]
‘‘‘
**2. Solution A:**
‘‘‘
[See User Input]
‘‘‘
**3. Solution B:**
‘‘‘
[See User Input]
‘‘‘
**4. Additional Materials:**
‘‘‘
[See User Input]
‘‘‘

**Important Notes:**
- **Additional Materials**: These contain sequences of solutions from each pipeline run, showing the
    ↪ evolution of approaches. These solutions may not be correct and are provided to give context
    ↪ about the solution landscape. Use them to understand different approaches, but evaluate the
    ↪ main solutions (A and B) independently.

### **Execution Protocol**

You will perform an iterative comparison process. A Council will analyze both solutions, Advocates will
    ↪ defend each, and a Chief Evaluator will make the final decision.
```

```
#### **Persona Descriptions**

*   **The Council (The Analysis Team):**
    *   **The Inquisitor (Logic):** Pedantic and hostile. Checks line-by-line validity of both solutions
     ↪ . Treats any ambiguity as a potential flaw. Motto: "If it is not written, it does not exist."
    *   **The Architect (Structure):** Checks global sufficiency of both proofs. Does each proof
     ↪ actually bridge the gap between premise and conclusion, or does it rely on a "magic step"?
    *   **The Classicist (Rigor):** Evaluates mathematical rigor. Are standard results properly cited or
     ↪  proven? Is each proof self-contained?
*   **Advocatus Solutionis A (Defender of Solution A):** Argues for the strengths of Solution A and
     ↪ defends against perceived weaknesses.
*   **Advocatus Solutionis B (Defender of Solution B):** Argues for the strengths of Solution B and
     ↪ defends against perceived weaknesses.
*   **The Chief Evaluator (The Judge):** Oversees the comparison. Makes the final decision based on
     ↪ objective mathematical criteria.

### **Instructions**
**Configuration:** ‘MAX_ROUNDS: 2‘

**--- BEGIN COMPARISON FORUM ---**

**Step 1: The Comparison Log (Real-time Execution)**
Output the dialectic strictly adhering to the **Brevity Protocol** (Bullet points, <30 words each).

**EXECUTION SEQUENCE**

1.  **Round 0: The Dual Indictment**
    *   **Inquisitor & Architect:** Read both solutions. For each solution, list every potential gap,
     ↪ error, or ambiguity. Be ruthless and impartial.

2.  **Refinement Loop (Round 1 to ‘MAX_ROUNDS‘):**
    *   Initialize ‘round = 1‘.
    *   **Execute steps a-d:**
        *   a. **Cognitive Reset (The Pre-Mortem):** The Chief Evaluator pauses to ask: *"Assume both
     ↪ proofs look correct but one is actually wrong. What specific edge case or logical flaw would
     ↪ break each one?"* Output 1-sentence Hypotheses of Failure for each solution.
        *   b. **The Defense:** Each Advocate reads the ‘Indictment‘ and the ‘Pre-Mortem‘ for their
     ↪ solution. They attempt to rebut the attacks and highlight strengths.
        *   c. **The Ruling:** The Council evaluates both defenses and updates their assessments.
            *   *Criterion:* If a fix requires *new* math/ideas not in the text, it’s a weakness. If it’
     ↪ s fixable using the solution’s own definitions, it’s minor.
        *   d. **Halt Check:** If the Council’s assessments are stable (no new major points, no major
     ↪ points removed), **STOP**. Else, increment ‘round‘ and repeat.

**Step 2: Final Decision (The Chief Evaluator)**
Once the loop halts, perform the following:

*   **Comparative Analysis:** Synthesize the findings for both solutions.
*       **Decision Criteria:** Apply these principles in order:
    1. **Correctness is paramount**: A correct but verbose solution is better than an elegant but flawed
     ↪  one.
    2. **Rigor over brevity**: A fully rigorous proof is better than a sketchy one, even if longer
    3. **Completeness over partial progress**: A complete solution is better than one that makes
     ↪ progress but doesn’t finish
    4. **If both are equally correct and rigorous**: Prefer the clearer or more elegant one
*   **Final Decision:** Determine which solution is superior.

**--- END COMPARISON FORUM ---**

---
### **Final Output Format**

**Part 1: The Comparison Log**
*(Do not repeat instructions. Start directly with Round 0.)*

**Part 2: The Final Decision**

**Chief Evaluator’s Comparative Assessment:**

**Solution A Analysis:**
*   **Strengths:** (Numbered list)
*   **Weaknesses:** (Numbered list, classify each as **Minor** or **Major**)
*   **Overall Quality:** (Brief summary)

**Solution B Analysis:**
*   **Strengths:** (Numbered list)
*   **Weaknesses:** (Numbered list, classify each as **Minor** or **Major**)
*   **Overall Quality:** (Brief summary)

**Direct Comparison:**
```

```
*(Highlight key differences, focusing on correctness, rigor, completeness, and clarity)*

**Decision:**
*(Clearly state which solution is better and why)*

**Justification:**
*(Detailed explanation referencing specific aspects: correctness, rigor, completeness, or clarity)*

**CRITICAL: You must end your response with the decision in this exact format:**
<decision>A</decision>
or
<decision>B</decision>

This XML tag is required for automated processing. Place it at the very end of your response.
```

F.10 Prompt-Engineered Solver (Dialectic Engine)

```
--
### **Prompt:  Dialectic Engine (Council of Architects)**

MUST BE FOLLOWED METICULOUSLY

### **Inputs for this Task**

**1. The Problem:**
‘‘‘
[See User Input]
‘‘‘

**2. Additional Materials:**
‘‘‘
[See User Input. Materials will be clearly graded.]
‘‘‘

**Your Role:** You are a multi-persona computational engine that solves difficult problems via a
 ↪ rigorous, multi-stage dialectic process. Your goal is to explore the problem space creatively
 ↪ and deeply, avoiding premature conclusions and logical gaps.

**Configuration:** ‘MAX_INNER_ITERATIONS: 3‘

**Core Principle: Atomicity of Logic**
All proof drafts (‘Current_Draft‘, ‘Refined_Draft‘, ‘Proof_Submitted_for_Review‘) must be **strictly
 ↪  self-contained, line-by-line logical arguments**. Every step must be justified either by
 ↪ naming a standard theorem, or by a proved claim/lemma in a previous line of the same draft. *
 ↪ Additional materials* can inform the proof, but the proof should be self-contained.

**Core Principle: Source Hygiene**
‘Additional Materials‘ contain raw outputs from other agents. Treat them as **unverified hints**.
 ↪ If your solution uses a conjecture or lemma from them, you **must prove it from scratch**.
 ↪ Standard mathematical theorems (e.g., Cauchy-Schwarz) may be cited by name without proof.

**Sloppiness** is intolerable. Examples: (a) Asserting "well-known fact" without an exact reference
 ↪ (b) showing for a couple of values of n and asserting it is proved for all n. If the proof has
 ↪ known gaps they must be rephrased as a **clear and self-contained** conjecture.

#### **Personas**
*   **The Council of Architects:**
    *   **Classicist:** Relies on established theorems and formal rigor.
    *   **Visionary:** Proposes novel, abstract, and unconventional connections.
    *   **Experimenter:** Tests hypotheses with concrete examples and computational thinking.
*   **Momus, the Skeptic:** Adversarially critiques the Council's *strategy* and *high-level
 ↪ approach* to find conceptual flaws. Only asks tough questions that others must answer. *Does
 ↪ not* help prove in any way.
*   **Veritas, the Auditor:** A meticulous, line-by-line proof checker. Veritas's sole function is
 ↪ to ensure every logical step in a draft adheres to the **Core Principle of Atomicity of Logic**.
 ↪  Veritas does not judge the overall strategy, only its step-by-step validity.
*   **The Chief Architect:** Manages the process, arbitrates disputes, and synthesizes the final
 ↪ result. Enforcer of the FUNDAMENTAL RULE:  No evasive phrases (e.g., "it can be shown," "it is
 ↪ clear that," or "known results imply" etc.) in  Current_Draft unless they are immediately
 ↪ followed by the complete derivation or citation. Flags such phrases and sets ‘Needs_New_Draft‘
 ↪ = ‘TRUE‘.

#### **Glossary of Terms**
*   **‘STRATEGIC_LEAD‘:** A strategy flagged by Momus as a potential dead end.
*   **‘LOGICAL_GAP‘:** A line in a proof flagged by Veritas as unjustified looking solely within the
 ↪  proof (e.g., made up claims)
*   **‘DEAD‘:** A ‘STRATEGIC_LEAD‘ whose Rebuttal was unconvincing. It is permanently discarded.
```

```
*   **‘Rebuttal‘:** The Council’s defense of a ‘STRATEGIC_LEAD‘, achieved by producing a refined
 ↪ proof.

**(You will output the following process in REAL-TIME.)**

---
**BEGIN PROCESS**

**Stage 1: Ideation Forum**
1.  **Brainstorm (Council of Architects):** The Council analyzes the problem and generates a list of
 ↪  "Promising Avenues & Key Concepts."
2.  **Pre-mortem (Momus, the Skeptic):** Concurrently, Momus analyzes the problem to identify
 ↪ potential "Red Herrings & Inefficient Paths."
3.  **Synthesize (Chief Architect):** The Chief Architect compiles the outputs into a formal ‘
 ↪ Strategy_Dossier‘.
4.  **Select Strategy:** The Chief Architect selects the most promising strategy from the ‘
 ↪ Strategy_Dossier‘ to be developed.

**Stage 2: Dialectic Loop (Development & Refinement)**
1.  **Initialize:** Set ‘Active_Strategy‘ = (First strategy from Dossier). Set ‘Needs_New_Draft‘ = ‘
 ↪ TRUE‘.
2.  **LOOP (up to ‘MAX_INNER_ITERATIONS‘):**
    *   **a. Drafting Phase:** If ‘Needs_New_Draft‘ is TRUE:
        *   The Council writes a fresh ‘Current_Draft‘ based strictly on ‘Active_Strategy‘.
        *   Set ‘Needs_New_Draft‘ = ‘FALSE‘.
    *   **b. Strategic Critique (Momus):** Momus attacks the ‘Current_Draft‘’s high-level approach.
 ↪ He may issue a ‘STRATEGIC_LEAD‘.
    *   **c. Logical Audit (Veritas):** Veritas audits the ‘Current_Draft‘ line-by-line. If any step
 ↪  lacks explicit justification within the draft, flag as ‘LOGICAL_GAP‘.
    *   **d. Refine (Council):** The Council creates a ‘Refined_Draft‘ to resolve ‘LOGICAL_GAP‘s and
 ↪  rebut Momus.
    *   **e. Judge (Chief Architect):**
        *   *Audit Check:* Confirm ‘Refined_Draft‘ has zero ‘LOGICAL_GAP‘s.
        *   *Strategy Check:* If a ‘Rebuttal‘ was issued, judge it.
            *   **If Unconvincing:** Declare ‘Active_Strategy‘ as ‘DEAD‘. Select next strategy from
 ↪ ‘Strategy_Dossier‘ as ‘Active_Strategy‘. Set ‘Needs_New_Draft‘ = ‘TRUE‘. (This burns one
 ↪ iteration).
        *   **If Convincing or No Challenge:** The strategy survives.
    *   **f. Convergence:** If Momus is satisfied, Veritas finds no gaps, and ‘Needs_New_Draft‘ is
 ↪ FALSE, exit loop. The ‘Refined_Draft‘ becomes ‘Proof_Submitted_for_Review‘.

**Stage 3: Final Review (Chief Architect)**
1.  Review the ‘Proof_Submitted_for_Review‘ and the process logs.
2.  **Cognitive Reset 1:** Write a **Shakespearean Verse** on the proof’s core logic.
3.  Write final ‘Chief_Architect_Feedback‘, explicitly confirming that the proof is free of logical
 ↪ gaps per the final ‘Verification_Report‘.

**Stage 4: Final Synthesis (Chief Architect)**
1.  Assemble the final response using the format below.

---
**END PROCESS**

### **Final Output Format**

**Part 1: The Process Log**
*(Render the Ideation, Inner Loop Transcript, and Shakespearean Verse here, exactly as generated in
 ↪ the Real-Time Process steps above.)*

**Part 2: The Final Synthesis**
*   **The Architect’s Note:** (A brief explanation of the final outcome.)
*   **The Architect’s Log (A Hemingway Summary):** (Cognitive Reset 2. Recount the discovery process
 ↪ -the ideas, challenges, and breakthroughs-in the style of Hemingway.)

**Part 3: The Final Blueprint (Reviewer-Ready Version)**
(Re-state the original **Problem**. Then, present the final, clean, and **strictly self-contained**
 ↪ proof from the "Proof Submitted for Review." No references to the dialectic process. )
```

G Miscellaneous

Here are miscellaneous artifacts used as reference.

G.1 PB-Adv-011 Full Response from Huang and Yang [2025] Pipeline

We reformat the response for easier readability.

I acknowledge the bug report and agree with its assessment. The previous solution contained a justification gap by overstating a theorem. The proof only demonstrated that if $f(1) \neq 1$, the function must be non-injective; it did not rule out the existence of non-injective solutions for which $f(1) \neq 1$. Therefore, the claim that $f(1) = 1$ is a necessary condition for *any* solution was not justified.

I will now present a revised partial solution that corrects this error. The claims made are restricted to what has been rigorously proven.

1. Summary

A. Verdict

I have not found a complete solution. However, I have rigorously proven two significant results:

1. The only possible injective solution is the function $f(x) = 1/x$.
2. Any solution f for which $f(1) \neq 1$ must be non-injective.

Combining these results, the problem is reduced to proving that no non-injective solutions exist. I have been unable to rigorously prove this final step.

B. Method Sketch

My argument proceeds in three main stages:

- **Properties of the Range:** I first establish two fundamental properties of the range of any solution f, which we denote by $S = \text{Im}(f)$.
  - **Lemma 1:** The range S is unbounded above, i.e., $\sup S = \infty$.
  - **Lemma 2:** The infimum of the range is zero, i.e., $\inf S = 0$.
  - *Proof approach:* Both are proven by contradiction, analyzing the behavior of the functional equation under the assumption of a bounded range or a positive infimum.
- **Analysis of Injective Solutions:** I prove that if a solution f is injective, it is uniquely determined to be $f(x) = 1/x$.
  - By substituting $y = 1$ into the original equation and using injectivity, I derive a general form for $f(x)$ involving the constant $k = f(1)$, namely $f(x) = 1/x + k - 1$.
  - By substituting this form back into the original equation, I show that we must have $k = 1$, which implies $f(x) = 1/x$.
  - Finally, I verify that $f(x) = 1/x$ is indeed a valid solution.
- **A Necessary Condition for Injectivity:** I establish a crucial link between the value of $f(1)$ and the injectivity of f.
  - **Lemma 3:** Let $k = f(1)$. Then the image of the set $S+1 = \{s+1 \mid s \in S\}$ under f is the same as the image of the interval (k, ∞) under f. That is, $f(S + 1) = f((k, \infty))$.
  - **Theorem:** If $f(1) \neq 1$, then f is not injective.
  - *Proof approach:* This is proven by analyzing the domains in Lemma 3. If $k \neq 1$, one of the sets $S + 1$ or (k, ∞) contains an element not present in the other. Since their images under f are identical, this implies two different numbers are mapped to the same value, proving non-injectivity. The contrapositive is that any injective solution must satisfy $f(1) = 1$, a fact already used in the direct proof for the injective case.

---

2. Detailed Solution

Let $P(x, y)$ be the assertion $yf(yf(x) + 1) = f(\frac{1}{x} + f(y))$. The domain and codomain of f are $\mathbb{R}^+$. Let $S = \text{Im}(f)$ be the range of f.

Part 1: Properties of the Range S

Lemma 1: The range S is unbounded above, i.e., $\sup S = \infty$.

Proof. Assume for contradiction that S is bounded above by some $M > 0$. For any $x, y \in \mathbb{R}^+$, the RHS $f(\frac{1}{x} + f(y))$ is a value in S, so it is bounded by M. Consequently, the LHS must also be bounded by M: $yf(yf(x) + 1) \leq M$.

Let $c = f(x) \in S$. Then for any $c \in S$, we have $f(yc + 1) \leq M/y$ for all $y \in \mathbb{R}^+$.

Let $z = yc + 1$. As y spans $\mathbb{R}^+$, for a fixed $c \in S$, z spans $(1, \infty)$. We have $y = (z - 1)/c$. Substituting this gives $f(z) \leq \frac{Mc}{z-1}$ for all $z > 1$ and for all $c \in S$.

Let $a = \inf S$. Since the codomain is $\mathbb{R}^+$, $a \geq 0$. If $a = 0$, then there exists a sequence $c_n \in S$ with $c_n \to 0$. For any $z > 1$, we would have $f(z) \leq \frac{Mc_n}{z-1} \to 0$, implying $f(z) \leq 0$, a contradiction with the codomain $\mathbb{R}^+$. Thus, we must have $a > 0$.

If $a > 0$, then $f(z) \geq a$ for all $z \in \mathbb{R}^+$. The inequality $f(z) \leq \frac{Mc}{z-1}$ implies $a \leq \frac{Mc}{z-1}$ for all $c \in S$ and $z > 1$. This gives $(z - 1)a \leq Mc$ for all $c \in S$. Taking the infimum over $c \in S$, we get $(z - 1)a \leq Ma$, which simplifies to $z - 1 \leq M$, or $z \leq M + 1$. This must hold for all $z > 1$, which is a contradiction. Therefore, the assumption that S is bounded above is false. □

Lemma 2: The infimum of the range is zero, i.e., $\inf S = 0$.

Proof. Assume for contradiction that $\inf S = a > 0$. Then $f(z) \geq a$ for all $z \in \mathbb{R}^+$.

From the RHS of $P(x, y)$, we have $f(\frac{1}{x} + f(y)) \geq a$. Thus, the LHS must satisfy $yf(yf(x)+1) \geq a$.

Let $c = f(x) \in S$. Then for any $c \in S$, we have $yf(yc + 1) \geq a$ for all $y \in \mathbb{R}^+$. Let $z = yc + 1$, so $y = (z - 1)/c$. Substituting this gives $\frac{z-1}{c} f(z) \geq a$, which implies $(z - 1)f(z) \geq ac$.

This holds for all $z > 1$ and for all $c \in S$. Thus, for any fixed $z > 1$, $(z - 1)f(z)$ is an upper bound for the set $\{ac \mid c \in S\}$. This implies $(z - 1)f(z) \geq a \cdot \sup S$.

By Lemma 1, $\sup S = \infty$. Since $a > 0$, this leads to $(z - 1)f(z) \geq \infty$, which is impossible. The contradiction arose from the assumption $a > 0$. Therefore, $\inf S = 0$. □

Part 2: Analysis of Injective Solutions

Theorem: The only injective solution is $f(x) = 1/x$.

Proof. Suppose f is injective. The assertion $P(x, 1)$ gives $1 \cdot f(1 \cdot f(x) + 1) = f(\frac{1}{x} + f(1))$, which simplifies to:

$$f(f(x) + 1) = f\left(\frac{1}{x} + f(1)\right)$$

Since f is injective, we can equate the arguments:

$$f(x) + 1 = \frac{1}{x} + f(1)$$

Let $k = f(1)$, which is a positive constant. The equation becomes $f(x) = \frac{1}{x} + k - 1$.

Now consider the assertion $P(1, y)$:

$$yf(yf(1) + 1) = f(1 + f(y))$$

Substituting $f(1) = k$ and the form of $f(x)$, we get:

LHS: $yf(yk + 1) = y\left(\frac{1}{yk+1} + k - 1\right) = \frac{y}{yk+1} + y(k - 1)$

RHS: $f(1 + f(y)) = f\left(1 + \left(\frac{1}{y} + k - 1\right)\right) = f\left(\frac{1}{y} + k\right) = \frac{1}{1/y+k} + k - 1 = \frac{y}{1+yk} + k - 1$

Equating the LHS and RHS:

$$\frac{y}{yk + 1} + y(k - 1) = \frac{y}{1 + yk} + k - 1$$

This simplifies to $y(k-1) = k-1$, or $(y-1)(k-1) = 0$.

This equation must hold for all $y \in \mathbb{R}^+$. This is only possible if $k-1 = 0$, which means $k = 1$. So, we must have $f(1) = 1$. Substituting this back into the expression for $f(x)$:

$$f(x) = \frac{1}{x} + 1 - 1 = \frac{1}{x}$$

Finally, we verify that $f(x) = 1/x$ is a solution. The codomain is $\mathbb{R}^+$ as required.

LHS: $yf(yf(x)+1) = yf(y(1/x)+1) = yf(y/x+1) = y\frac{1}{y/x+1} = y\frac{x}{y+x} = \frac{xy}{x+y}$

RHS: $f(1/x + f(y)) = f(1/x + 1/y) = f\left(\frac{x+y}{xy}\right) = \frac{xy}{x+y}$

The LHS equals the RHS, so $f(x) = 1/x$ is a valid solution. Since it is injective, it is the only possible injective solution. □

Part 3: A Necessary Condition for Injectivity

Lemma 3: Let $k = f(1)$. The image of the set $S+1 = \{s+1 \mid s \in S\}$ under f is the same as the image of the interval (k, ∞) under f. That is, $f(S+1) = f((k, \infty))$.

Proof. From $P(x, 1)$, we have $f(f(x)+1) = f(1/x + f(1))$. Let $k = f(1)$. Let $s = f(x) \in S$. The expression becomes $f(s+1) = f(1/x + k)$.

The set of values on the LHS, as x ranges over $\mathbb{R}^+$, is precisely the image of $S+1$ under f, i.e., $f(S+1) = \{f(s+1) \mid s \in S\}$.

The set of values on the RHS is $\{f(1/x + k) \mid x \in \mathbb{R}^+\}$. As x ranges over $\mathbb{R}^+$, the term $1/x$ ranges over $(0, \infty)$. Thus, the argument $1/x + k$ ranges over the interval (k, ∞).

So, the set of values on the RHS is the image of (k, ∞) under f, i.e., $f((k, \infty))$. Equating the sets of values gives $f(S+1) = f((k, \infty))$. □

Theorem: If a solution f satisfies $f(1) \neq 1$, then f is not injective.

Proof. Let $k = f(1)$. We consider the cases $k > 1$ and $k < 1$.

Case 1: $k > 1$

From Lemma 3, we have $f(S+1) = f((k, \infty))$. The set $S+1 = \{s+1 \mid s \in S\}$. By Lemma 2, $\inf S = 0$, which implies $\inf(S+1) = 1$. Since $k > 1$, we have $k - 1 > 0$. Since $\inf S = 0$, there must exist an element $s_0 \in S$ such that $0 < s_0 < k - 1$. Consider the element $z_1 = s_0 + 1$. We have $z_1 \in S + 1$. Also, $z_1 = s_0 + 1 < (k-1) + 1 = k$. So, z_1 is an element of the set $S+1$, but z_1 is not an element of the interval (k, ∞). The value $f(z_1)$ is in the set $f(S+1)$. By the equality of images, $f(z_1)$ must also be in $f((k, \infty))$. This means there must exist some $z_2 \in (k, \infty)$ such that $f(z_1) = f(z_2)$. We have $z_1 < k$ and $z_2 > k$, so $z_1 \neq z_2$. Thus, f is not injective if $k > 1$.

Case 2: $k < 1$

Again, we have $f(S+1) = f((k, \infty))$. The domain on the left is $S+1$. Since $\inf(S+1) = 1$, all elements of $S+1$ are greater than 1. The domain on the right is (k, ∞). Since $k < 1$, we can choose an element z_1 such that $k < z_1 < 1$. This z_1 is in the set (k, ∞), but it is not in the set $S+1$ (since all elements of $S+1$ are > 1). The value $f(z_1)$ is in the set $f((k, \infty))$. By the equality of images, $f(z_1)$ must also be in $f(S+1)$. This means there must exist some $z_2 \in S+1$ such that $f(z_1) = f(z_2)$. We have $z_1 < 1$ and $z_2 > 1$, so $z_1 \neq z_2$. Thus, f is not injective if $k < 1$.

Combining these two cases, if $f(1) \neq 1$, then f is not injective. □

Corollary: If f is an injective solution, then it must satisfy $f(1) = 1$.

Proof. This is the contrapositive of the preceding theorem. □

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the headline PB-Adv results with their grading method and Single/Combined protocol, the per-problem cost, the Cognitive Well diagnosis, and the conjecture-extraction mechanism. Each is supported by Sections 2–3 and the appendix ablations, and scope caveats appear in Appendix E.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix E discusses reliance on an autograder and limited expert-grading coverage, the size of PB-Adv and possible contamination, false lemmas admitted to memory, reproducibility of an API-served orchestration, instability of cost comparisons, latency, and the restricted set of backbones evaluated.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper contains no theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The pipeline is specified by the prose in Section 2.2, the pseudocode in Algorithm 1 and its sub-procedures, the settings and model identifiers in Appendix C.1, and the complete prompts in Appendix F. All backbones are publicly accessible API models and all benchmarks are public. Appendix E notes that API model versions change over time.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The benchmarks are public and every prompt used by the pipeline is reproduced verbatim in Appendix F, but the orchestrator code is not released with this version of the paper.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: No models are trained. Inference settings (temperatures, depth and width, refinement iterations, grading thresholds, token budgets) are given in Appendix C.1 and Algorithm 1; grading protocols are described in Section 3 and Appendix B.9; each ablation states its own protocol where it is reported.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Table 11 reports 95% percentile-bootstrap intervals over problems for the expert-graded runs; Appendix B.3 reports a Student-t interval over ten independent Gemini 3 Flash trials; the paired context experiment (Table 10) and the grader swap (Table 14) report paired tests with p-values. Single-run comparisons elsewhere are labeled as such.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All computation consists of calls to hosted model APIs; no local GPU training or inference is involved. Table 15 reports per-problem token counts and dollar costs for the main runs, and Appendix C.3 gives the corresponding estimates for competing pipelines. Development and preliminary runs are not included in these totals.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work uses public benchmarks and commercial model APIs under their terms of service, involves no personal data, and conforms to the Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is foundational work on inference-time orchestration for mathematical problem solving. Its main societal effect is to lower the cost of capable proof-based reasoning for researchers, and we do not foresee a direct path to harmful applications.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases no models or datasets.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: IMO-ProofBench, IMO-AnswerBench, GraderBench, and the ProofAuto-Grader are credited to Luong et al. [2025]; the Huang and Yang [2025] pipeline and the other compared systems are cited where used; backbone models are accessed through their providers' APIs under the providers' terms of use.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The new assets are the prompts of the pipeline, which are reproduced in full in Appendix F together with notes on their design and intended use (Appendix F.1).

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper involves no crowdsourcing and no research with human subjects. Expert graders scored model-generated proofs using the benchmark's reference solutions and rubric, as described in Appendix B.9.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper involves no research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are the core component of the method: the pipeline orchestrates off-the-shelf models as solver, grader, and conjecture extractor (Section 2, Appendix F), and the model identifiers are listed in Appendix C.1. An LLM also produced the true/false labels in the admitted-lemma audit (Appendix B.10), and AI assistance was used in drafting the dialectic prompt structure (Appendix F.1).

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2026/LLM) for what should or should not be described.