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

Xingyu Dang [*1] **Rohit Agarwal** [*1] **Rodrigo Porto** [1] **Anirudh Goyal** **Liam Fowl** [*2] **Sanjeev Arora** [2]

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% performance using Gemini 3.1 Pro with an average cost per question of $\sim$ $9. 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[1].

[*]Equal contribution [1]Princeton University, New Jersey, USA [2]Princeton Language and Intelligence. Correspondence to: Sanjeev Arora <arora@cs.princeton.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026. Copyright 2026 by the author(s).

[1]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 & Yang (2025) at 24% expert-graded accuracy.

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 & 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 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.

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 & 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

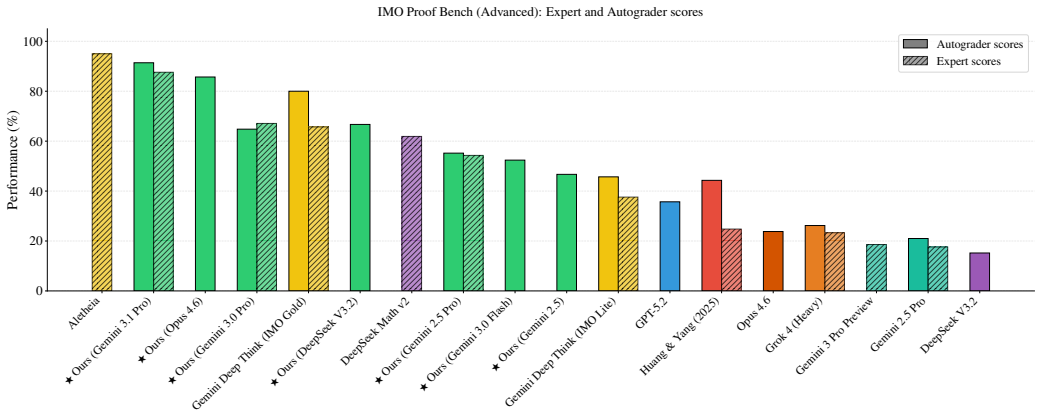

Figure 1. Results for our proposed method versus competing methods and models. Both expert graded and autograded results included (where possible). Our proposed method outperforms all open-source pipelines, while being significantly cheaper than these pipelines (cf. Figure 3) and ranks second only to 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).

61.9% but, as noted, allows up to 3000 USD/question in API cost. The Huang & 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 1).

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 & 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[2]. We conjecture that such 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 & Yang (2025) grader in Appendix D.

Appendix H presents examples that illustrate how our method circumvents the above failure modes. The improvements demonstrated in Figures 1 and 3 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 3.5.

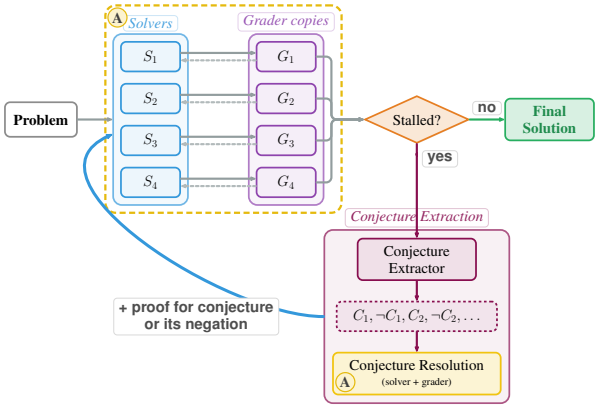

Figure 2. A high level overview of our pipeline. A problem is processed by parallel solver–grader pairs (module **A**). When progress stalls, the *Conjecture Extractor* identifies candidate lemmas C_i and their negations from current proof attempts. Each conjecture is resolved by an independent solver–grader module (also **A**), with proofs (for either the conjecture or its negation) fed back to enable further progress toward the final solution.

[2]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.

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[3] seems easier than **finding** a correct proof[4], 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 & 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 accurate/helpful graders for within a math-solving pipeline, as well as new suggestions for how to evaluate graders (cf. Section 3.5).
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 Pro/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 I) 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 C and Table B), but overall, we stress the main benefit is convenience and precise control[5].

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) in-

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

[4]The intuition is formalized as the famous $P \neq NP$ conjecture (Arora & Barak, 2009).

[5]Basic prompting methdology was hand-designed using AI help before release of PB-Adv, using Gemini 2.5 pro, 6 IMO2025 questions and another 5-6 questions from `artofproblemsolving.com`. Due to budget constraints, no automated methods were used for prompt design. It was adapted to Gemini 3 pro using help from the model itself. We refer to our pipeline as "Momus pipeline," with Momus being a picky persona that was named after a mythical figure in Greek philosophy.

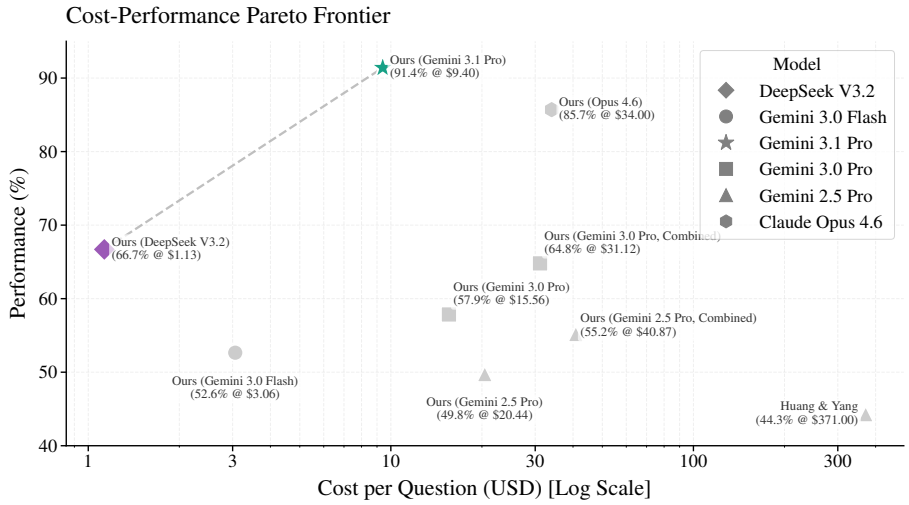

Figure 3. Pareto frontier of cost vs. performance measured using the autograder in Luong et al. (2025). "Single" denotes that the output of a *single* run of the pipeline. "Combined" denotes our standard algorithm for PB-Adv which uses a further model call to choose from two parallel pipeline runs - cf. Section 2.2 "Post Enhancement and Aggregation"). All costs calculated with January 2026 pricing.

curs 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.

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*. We say a candidate solution is *plausible but unverified* when it has scored highly in one or more internal grading rounds (so it appears acceptable to the system) but has failed to reach the strict consensus threshold of $7/7$ from three independent grading runs that we use to terminate refinement. For such solutions—and equivalently for any candidate stuck at a score of $\leq 6/7$—we assume the verifier is "contaminated" by the parent proof's context 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 instantiate two independent copies of the Phase I procedure in parallel to prove both C_j and its negation $\neg C_j$.

4. **Resolution:** If one of $C_j, \neg C_j$ is proven, it is added to a persistent global memory ($\mathcal{M}_{lemma}$) as a proven lemma. (conjecture or negation)

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 any partial progress on proving conjectures in the previous round, as this may be helpful to a solver. 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. If any of the 2 parallel session's conjectures have been fully proved, we include the corresponding written proof in $\mathcal{M}_{lemma}$ and launch a session of Phase III for the final solving.

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 6, Appendix F).

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 Figure 2.

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 3.5) 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. To establish that the pipeline is not specific to the Gemini family, we additionally report results with two further backbones (Claude 4.6 Opus and the open-weight DeepSeek-V3.2) in Section 3.2.

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 and Gemini 3 pro. We found high consistency between these (blind) expert scores and autograder scores, which suggests our pipeline may be subtly different from past works where a significant gap existed between the two grading methods (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 more than ten 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 & 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.

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 6 (Appendix F). 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.

3.1. Cost Comparison

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 1. Compared to straightforward calculations and implementations of other methods (see Appendix G.4), our solver system saves close to 90% of the total cost compared to the next closest Huang & 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 3 that our method dominates the Pareto frontier of advanced questions –demonstrating capable performance for as little as 3 USD/question!

Table 1. 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**

3.2. Generalization Across Backbones

The Gemini family powers both the leading academic pipeline of Huang & 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 and the open-weight DeepSeek-V3.2 as the backbone for all roles (solver, grader, and conjecture-extractor). Table 2 reports the autograded average score on PB-Advanced for each backbone alongside the score of a single base call to the same model.

Across both 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.

Table 2. 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.

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

3.3. Performance on Additional Benchmarks

Beyond PB-Adv (30 questions), we 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 final-answer questions). It outperforms every QED-Nano (LM-Provers et al., 2026) baseline on both, scoring 80.3% and 91.3% respectively despite using our weakest backbone (full results in Table 8, Appendix F).

3.4. Statistical Significance

To bound prover-sampling variance, 10 independent Gemini 3.0 Flash trials on PB-Advanced yield a Student-t 95% confidence interval of $[49.1\%, 56.1\%]$ on the mean autograded score; even its lower bound exceeds the autograded score of Huang & Yang (2025) at an orders-of-magnitude smaller budget (details in Appendix F.2).

3.5. 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. Thus our pipeline mostly ignores the numerical grade except for tie-breaking

and exiting early due to reaching a perfect score (with zero slips found in the proof) 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 3. 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 3 shows Momus with pipeline solutions is competitive with the best graders.

Does GraderBench performance capture pipeline usefulness? Not fully: vanilla Momus looks worse than Momus-w/-p.sols on GraderBench yet performs better *in* our pipeline. What matters for a pipeline (especially at low parallelism)—treating a score ≥ 6 as *positive* and ≤ 5 as *negative*[6]—is the **False Positive Rate** (FPR $= P(\text{pred} \geq 6 \mid \text{human} \leq 5)$), a critical "Cognitive Well" failure that risks submitting a wrong proof, versus the **False Negative Rate** (FNR $= P(\text{pred} \leq 5 \mid \text{human} \geq 6)$), where over-strictness merely triggers another (often improving) round. Table 4 reports these (full confusion matrices in Figure 7).

Table 4. Comparison of Momus Prompts vs. ProofAutoGrader (with reference solution) on GraderBench (1000 datapoints). MAE is expressed as a percentage of the maximum score of 7.

Method	**FPR** $\downarrow$	**FNR** $\downarrow$	**Acc** $\uparrow$	**MAE** $\downarrow$
Momus	17.2%	26.1%	28.1%	25.9%
ProofAutoGrader	31.7%	6.2%	59.4%	17.6%
Momus w/ p.sols	26.9%	8.8%	49.8%	19.9%

Momus's low FPR reflects a deliberate safety margin: pickiness mitigates Cognitive Wells at the cost of higher FNR, which is acceptable since the pipeline always returns the best solution found. Strikingly, even the gold ProofAutoGrader may be suboptimal *inside* a pipeline—its higher accuracy comes with a worse FPR that would tend to terminate on incorrect proofs. Momus-w/-p.sols is a reasonable middle ground (recovering much of the accuracy and nearly matching the MAE), but its higher FPR already appears to hurt pipeline utility. In short, a grader's value as a standalone scorer (GraderBench) need not align with its value as a feedback source in a pipeline—a tension, and the need for better metrics to capture it, that we leave to future work.

3.6. 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. With conjectures, the ProofAutoGrader scores increase monotonically, whereas without them they jump erratically or flatline; representative per-problem trajectories (including a PB-Adv-017 walkthrough) and a broader selection of runs appear in Figure 4 and Appendix E.

Aggregate effect at matched compute. The per-problem trajectories (Figure 4, Appendix E) 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

[6] If we combine the 0 and 1–3 buckets, the classification accuracies in Table 4 become more comparable: ProofAutoGrader 73.5%, Momus-w/-p.sols 72.0%, base Momus 53.7%.

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 confirms that the gains we attribute to conjecture extraction and contextual detachment are not just an artefact of additional solver diversity or sampling budget.

3.7. 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. To test this directly, we ran a controlled ablation on the conjecture-proving step. We took the lemmas extracted by our pipeline and tasked the prover with proving them under two conditions: (a) in isolation (the deployed setting), and (b) with the original problem context appended to the prompt (together with the fact that the lemma is purported to help solve the problem).

Introducing the original problem context dropped the lemma-proving success rate by 67% relative to the isolated setting. This is direct empirical evidence that the same model is significantly more capable on a sub-claim once it is detached from the contaminated parent context, and supports the contextual-detachment design of our pipeline.

4. Limitations

Several limitations warrant discussion. **(i)** Consistent with prior work (Luong et al., 2025), many of our ablations and comparisons rely on the autograder; although Luong et al. (2025) report high autograder–human correlation (basic $r = 0.98$, advanced $r = 0.93$), they also document occasional unreliability (e.g., a 15% gap on Deep Think (IMO Gold)). We therefore used an expert human grader (with gold solution and rubric) for two of our most important runs and found their grades consistent with the autograder. **(ii)** Evaluation data of sufficient size and contamination-free complexity is scarce: IMO-ProofBench-Advanced has only 30 problems, and we mitigate contamination by testing on backbones largely released *before* IMO-Bench. **(iii)** Our pipeline wins on cost and accuracy but loses to massively parallel pipelines on latency. **(iv)** Cost considerations prevented evaluating a wider set of frontier models and extensive prompt tuning.

5. Conclusion

Our work suggests that reliable LLM theorem proving may not require massive parallelism or thinking budgets: our efficient pipeline lets off-the-shelf models reach comparable performance at a fraction of the cost of published methods. Central to this is identifying the *Cognitive Plateau* and *Cognitive Well*—failure modes in which solver–grader pipelines become trapped in plausible but incorrect reasoning that fools their own internal verifiers—and circumventing them via conjecture extraction with contextual detachment, which dynamically isolates and resolves the load-bearing claims of a stalled proof. We suspect these failure modes are deep-rooted in current solver/grader pipelines and may extend to broader AI-for-Science systems, warranting further attention along with better grader evaluation.

On IMO-ProofBench-Advanced we scored (at the time of the experiments) ahead of all released competing methods, with expert-verified performance, and we show where a grader's value as a standalone scorer diverges from its role as a feedback provider—a tension we believe deserves further study as these pipelines mature.

References

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

Balunović, M., Dekoninck, J., Petrov, I., Jovanović, N., and Vechev, M. Matharena: Evaluating llms on uncontaminated math competitions. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark*, 2025.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., 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, July 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.

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

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025a.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025b.

Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., et al. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

LM-Provers, Qu, Y., Setlur, A., Dekoninck, J., Beeching, E., Li, J., Wu, I., Tunstall, L., and Kumar, A. Qed-nano: Teaching a tiny model to prove hard theorems. https://huggingface.co/spaces/lm-provers/qed-nano-blogpost, 2026. Blog post.

Luong, M.-T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35406–35430, 2025.

Luong, T. and Mirrokni, V. 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.

Madaan, L., Didolkar, A., Gururangan, S., Quan, J., Silva, R., Salakhutdinov, R., Zaheer, M., Arora, S., and Goyal, A. Rethinking thinking tokens: LLMs as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025.

Moura, L. d. and Ullrich, S. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pp. 625–635. Springer, 2021.

Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., et al. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Shao, Z., Luo, Y., Lu, C., Ren, Z., Hu, J., Ye, T., Gou, Z., Ma, S., and Zhang, X. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

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

A. Algorithm Details

Algorithm 1 is our main pipeline algorithm. Sub-functions are defined in other algorithms.

There are further details on sub-functions in Algorithm 2. Note that this is a pseudocode representing the logic in the natural language prompt.

B. Prompting framework ("Momus")

Dialectic-based prompting: Current pipelines (e.g. Huang & 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 & 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 Pro.

C. Momus Solver Evaluation: Single Call

To isolate the effect of prompt engineering, we compare two solver configurations:

- **Momus Solver**: A single call to our dialectic-based solver prompt. This is captured in Appendix I.9.[7]
- **Huang Yang Solver**: A single call using using the Solver prompt in Huang & 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 5 details the head-to-head performance.

Table 5. Comparison of one call to Momus Solver vs. Huang Yang Solver using Average Scores on Proofbench-Basic (30 problems, 4 attempts each). "Pass @ 1" indicates the score in the first attempt; "Pass @ 4" indicates the highest score across all four attempts.

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

The Momus solver achieves an average-of-4 score of $5.10/7$ versus $4.92/7$ for the Huang–Yang solver—a modest difference, indicating the dialectic prompt does not by itself yield a significantly better solver. This parity suggests that the performance gap in Figure 1 stems primarily from our pipeline's other innovations—grading, conjecture extraction, and contextual detachment—rather than the initial solver prompt; our dialectic methodology was used to design those roles as well.

D. Huang & Yang (2025) Grader Analysis

Due to resource constraints, we were unable to run the expensive Huang & 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 6.

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

We study this case further in Table 7 by running 20 independent verifications with the Huang & Yang (2025) verifier on the following solutions:

[7]This is a slight variant of the prompt in Appendix I.1.

Table 6. The counts of how many questions received each possible autograder result on Huang & 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 solution output by the Huang & 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."

Table 7. Huang & Yang Verifier run 20 times independently on the solution its 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	$0/20$

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.

E. Conjecture Removal: Further Analysis

We first isolate problems where the Gemini 2.5 Pro pipeline succeeds *and* proves at least one conjecture, then re-run them with the conjecture stage removed under a matched token budget, sampling 5 evenly spaced ProofAutoGrader checkpoints. Figure 4 shows two representative problems. On PB-Adv-017, neither setup makes progress on ProofAutoGrader grades for the first 60% of the run; in the last 40%, proven conjectures accumulate and let the model reach a perfect $7/7$. Without proved conjectures, the pipeline grader instead awards $6/7$ to a completely incorrect solution, getting stuck in the "Cognitive Well" and unable to improve.

Figure 5 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 5 ended up with a $0/1$ final score. 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.

F. Additional Quantitative Results

F.1. Performance on Additional Benchmarks

To address the concern that PB-Adv (30 questions) is a narrow basis for evaluation, 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). Table 8 reports the results.

Table 8. 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.

F.2. 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 & Yang (2025), despite our pipeline using a weaker backbone and a budget orders of magnitude smaller.

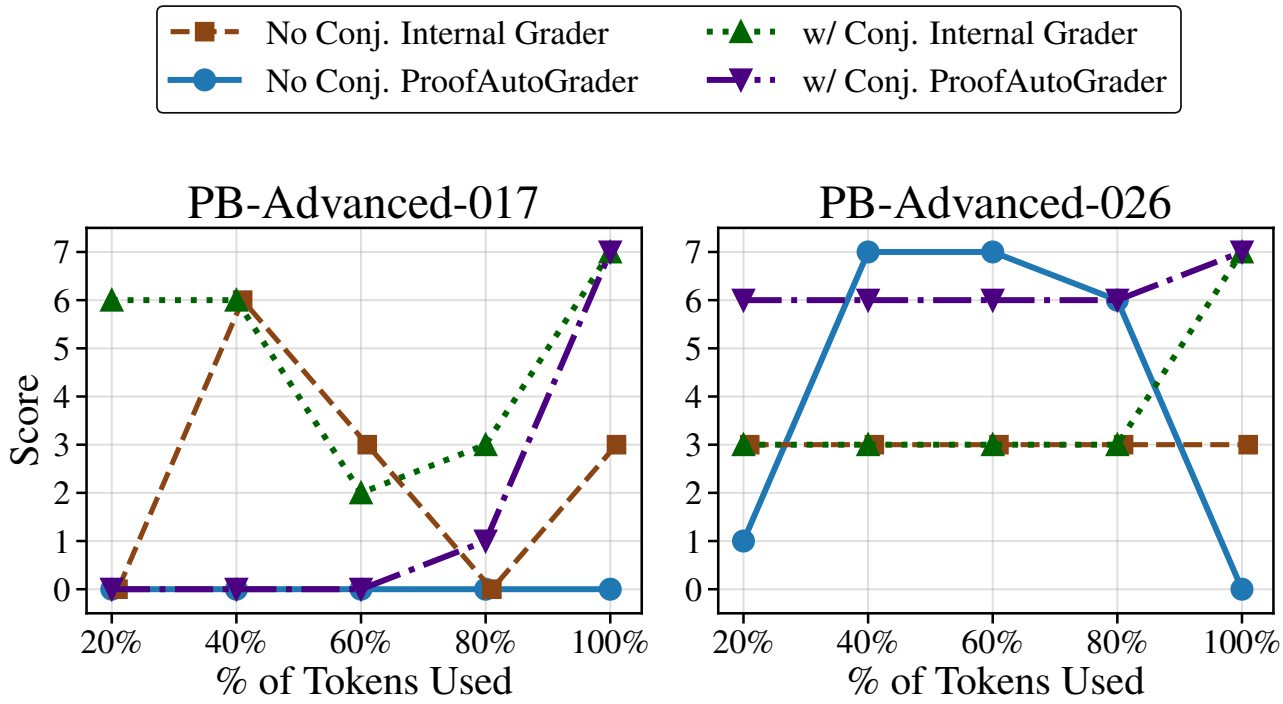

Figure 4. Study of benefits of conjecture extraction/resolution on the solution pipeline. We run (a) the pipeline without any conjecture steps and (b) with conjecture steps. For both runs, we equalize total budgets. We sample solutions for five phases throughout the runs. We then grade these runs using both the pipeline's internal grader and the external ProofAutoGrader. We see that with conjectures, the ProofAutoGrader scores monotonically increase, while those without conjectures may jump wildly or flatline. Upward spikes in conjecture-based solving correspond to proof/disproof of conjectures.

G. Compute details

G.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.

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.

G.2. 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 9 and 10 report autograded average scores.

Table 9. 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 10. 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

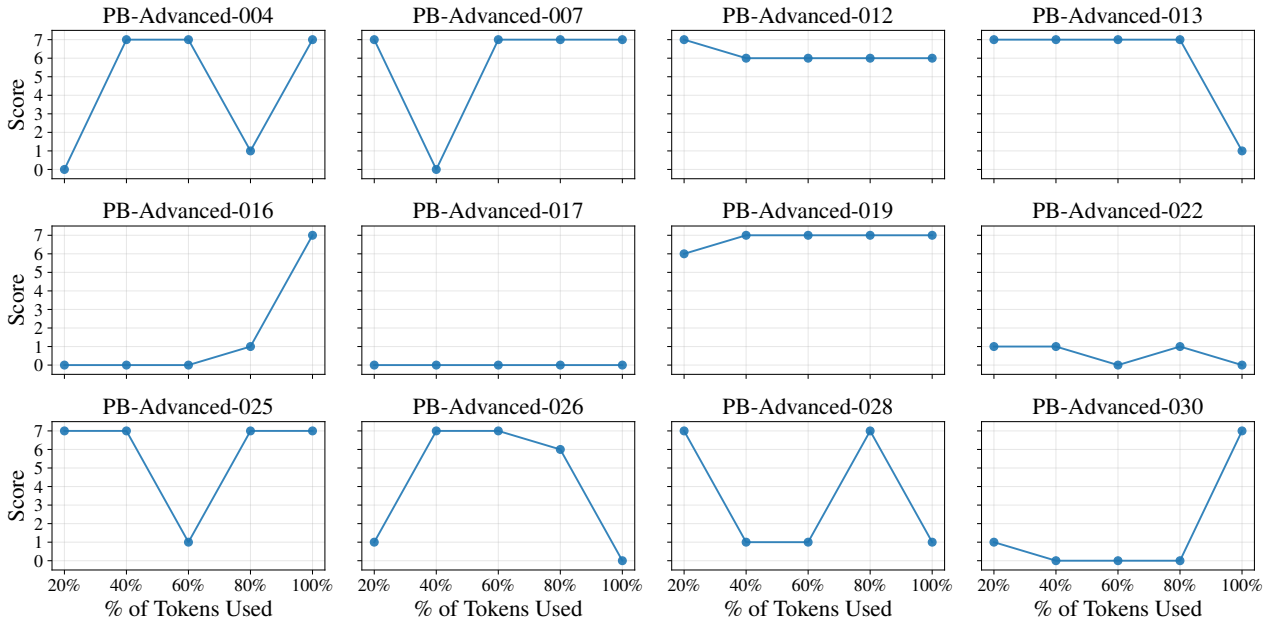

Figure 5. Effect of removing conjectures from successful runs of the solver pipeline, as described in Section 3.6. The x-axis denotes progression in time and the y-axis corresponds to the grade provided by ProofAutoGrader to the pipeline's best solution. After removal of conjectures, the solution process becomes erratic or confused. Compare to Figure 4

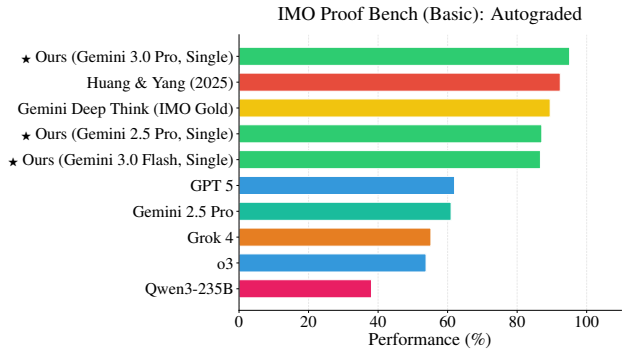

Figure 6. Results for IMO Proof Bench (Basic) according to the IMO-Bench autograder. "Single" here denotes that the output of a *single* run of the pipeline was submitted (as opposed to our standard algorithm for PB-Adv which uses a further model call to choose from two parallel pipeline runs - cf. Section 2.2 "Post Enhancement and Aggregation"). All comparison results taken from Luong et al. (2025).

PB-Advanced, suggesting that the pipeline is fairly robust to these hyperparameter choices and does not require fine grid-tuning to achieve strong performance.

G.3. Our Method

In Table 1 present more detailed analysis of the compute used to achieve our performance numbers on IMO-ProofBench Advanced.

G.4. 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.

For Huang & 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 "self-verification" 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.

H. 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.

H.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.

H.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 $L - U - V$.

H.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 further obtain solves this through

I. System Prompts

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).

I.1. Solver (Dialectic Engine)

Solver (Dialectic Engine)

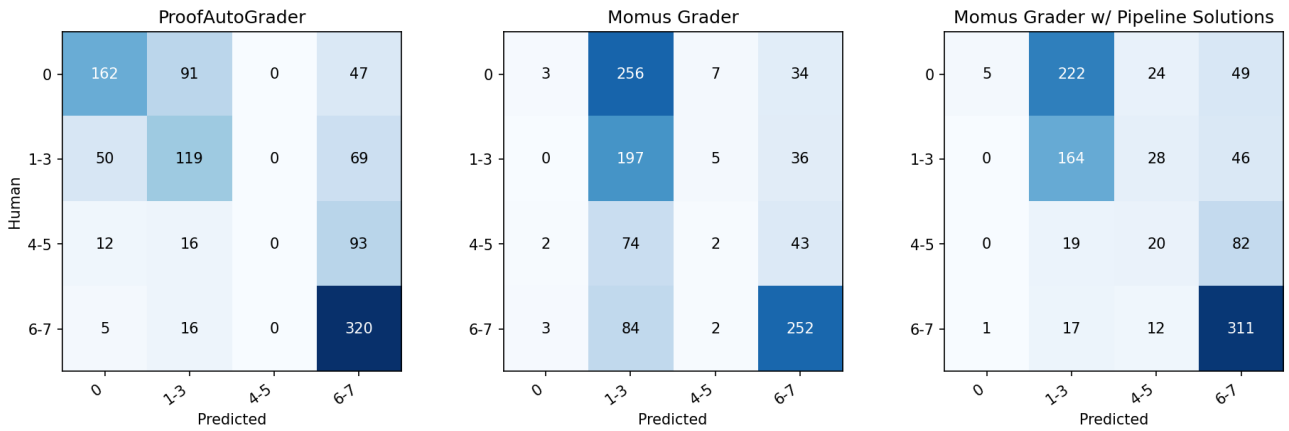

Figure 7. Confusion matrices of the two Momus graders against the autograder in Luong et al. (2025).

```
## 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.)*
```

I.2. 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.)
```

I.3. 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.)*
```

I.4. 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).
```

I.5. 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.)*
```

I.6. 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".
```

I.7. 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
```

I.8. 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.
```

I.9. 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. )
```

J. PB-Adv-011 Full Response from (Huang & 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. □

Algorithm 1 Memory-Augmented Dialectic Reasoning (Main Pipeline)

Input: Problem P, Initial Iterations L_0, Conjecture Iterations L, Solvers K, Threshold $\tau = 7$, Enhancement Threshold $\tau_e = 6$
Initialize: Solution Memory $\mathcal{M}_{sol} \leftarrow \emptyset$, Lemma Memory $\mathcal{M}_{lemma} \leftarrow \emptyset$, 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**
    $Ctx_k \leftarrow$ SELECTKTHTOP$(\mathcal{S}_{base}, k)$
  end for
  $\mathcal{S}_{base} \leftarrow \mathcal{S}_{base} \cup$ DIALECTICSOLVE$(P, \emptyset) \cup \bigcup_{k \leq K-1}$ DIALECTICSOLVE(P, Ctx_k)
  if VERIFIEDSUCCESS$(\mathcal{S}_{base}, N = 3)$ **then**
    return BEST$(\mathcal{S}_{base})$
  end if
end for
$\mathcal{M}_{sol} \leftarrow$ UPDATE$(\mathcal{M}_{sol}, \mathcal{S}_{base})$
for $l = 1$ **to** L **do**
  // PHASE 2: LEARNING FROM FAILURE
  $S_{seeds} \leftarrow$ SELECTTOP$(\mathcal{M}_{sol}, N = 2)$
  $\mathcal{H} \leftarrow$ EXTRACTHYPOTHESES$(S_{seeds}, \mathcal{M}_{lemma}, \mathcal{C}_{fail})$
  $\mathcal{L}_{new}, \mathcal{F}_{new} \leftarrow$ VERIFYHYPOTHESES$(\mathcal{H}, \tau)$
  $\mathcal{M}_{lemma} \leftarrow \mathcal{M}_{lemma} \cup \mathcal{L}_{new}$
  $\mathcal{C}_{fail} \leftarrow \mathcal{C}_{fail} \cup \mathcal{F}_{new}$
  // PHASE 3: SOLVING WITH ENHANCED CONTEXT
  $Ctx \leftarrow \mathcal{M}_{lemma} \cup$ RETRIEVEBEST$(\mathcal{M}_{sol})$
  $\mathcal{S}_{guided} \leftarrow$ DIALECTICSOLVE$(P, Ctx, K - 1)$
  $\mathcal{S}_{fresh} \leftarrow$ DIALECTICSOLVE$(P, \emptyset, 1)$ {Diversity check}
  $\mathcal{S}_{final} \leftarrow \mathcal{S}_{guided} \cup \mathcal{S}_{fresh}$
  $\mathcal{M}_{sol} \leftarrow$ UPDATE$(\mathcal{M}_{sol}, \mathcal{S}_{final})$
  if VERIFIEDSUCCESS$(\mathcal{S}_{final}, N = 3)$ **then**
    return BEST$(\mathcal{S}_{final})$
  end if
end for
// PHASE 4: POST-ENHANCEMENT FOR NEAR-PERFECT SOLUTIONS
$S^* \leftarrow$ BEST$(\mathcal{M}_{sol})$
$\mathcal{G}_{check} \leftarrow$ GRADEINDEPENDENT$(S^*, N = 3)$
if not ALLPERFECT$(\mathcal{G}_{check})$ **then**
  $\mathcal{H}_{gaps} \leftarrow$ EXTRACTGAPS$(S^*, \mathcal{G}_{check})$
  $\mathcal{L}_{fix}, \mathcal{F}_{fix} \leftarrow$ VERIFYHYPOTHESES$(\mathcal{H}_{gaps}, \tau_e)$
  $S_{better} \leftarrow$ DIALECTICSOLVE$(P, \mathcal{L}_{fix} \cup \mathcal{F}_{fix}, K)$
  $\mathcal{G}_{better} \leftarrow$ GRADEINDEPENDENT$(S_{better}, N = 3)$
  return argmax$\{$MEAN$(\mathcal{G}_{check})$, MEAN$(\mathcal{G}_{better})\}$
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
  $F'_i \leftarrow \text{LLM}_{\text{grader}}(S'_i)$
  $\mathcal{S}_{out} \leftarrow \mathcal{S}_{out} \cup \{S'_i\}$
end for
return $\mathcal{S}_{out}$

Algorithm 3 Function: VerifyHypotheses

Input: Candidates $\mathcal{H}$, Threshold τ
$\mathcal{L}_{proven} \leftarrow \emptyset, \quad \mathcal{F}_{failed} \leftarrow \emptyset$
for all $(C, \neg C) \in \mathcal{H}$ **do**
  $g_{pos} \leftarrow \text{EVAL}(\text{LLM}_{\text{solve}}(C))$
  $g_{neg} \leftarrow \text{EVAL}(\text{LLM}_{\text{solve}}(\neg C))$
  if $g_{pos} \geq \tau \wedge g_{neg} < \tau$ **then**
    $\mathcal{L}_{proven} \leftarrow \mathcal{L}_{proven} \cup \{C\}$
  else if $g_{neg} \geq \tau \wedge g_{pos} < \tau$ **then**
    $\mathcal{L}_{proven} \leftarrow \mathcal{L}_{proven} \cup \{\neg C\}$
  else
    $\mathcal{F}_{failed} \leftarrow \mathcal{F}_{failed} \cup \{(C, \text{Ambiguous})\}$
  end if
end for
return $\mathcal{L}_{proven}, \mathcal{F}_{failed}$

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$(S_1, \ldots, S_N)$