LeanRefiner: Agentic Global-to-Local Optimization of Lean Proofs

Anonymous ACL submission

Abstract

Modern Lean provers are increasingly able to produce proofs that pass mechanical verification. However, a verified proof may still be lengthy, repetitive, and difficult to read or maintain. We formalize this setting as *proof optimization after correctness* and propose **LeanRefiner**, a framework for optimizing verified Lean proofs under hard correctness constraints. Given a verified Lean proof, LeanRefiner aims to shorten and reorganize it while preserving the theorem statement, maintaining Lean verifiability, and avoiding proof holes. To achieve this goal, LeanRefiner employs a verifier-guided agentic system that combines theorem-level global restructuring with pattern-guided local reduction over bounded proof fragments. Experiments on three collections of verified Lean proofs show that LeanRefiner consistently reduces proof length while satisfying strict verifier constraints. Further analysis shows that these gains mainly stem from the staged collaboration between the global restructuring and local reduction agents, rather than from superficial cleanup or any specific backend.

1 Introduction

In formalized mathematics, mathematical statements and proofs are written in formal languages that can be checked by proof assistants (Harrison, 1996). This machine-checkable setting provides a rigorous basis for verifying both mathematical results and their supporting arguments. Lean and Mathlib (Moura and Ullrich, 2021) constitute one of the most active ecosystems for machine-checkable mathematics, combining a formal language, an interactive theorem prover, and a large mathematical library. Recent advances have made Lean-based theorem proving increasingly practical. By integrating retrieval-based guidance with interactive Lean feedback, automated proving systems can now generate verified proofs at increasingly larger scales (Yang et al., 2023; Zheng et al., 2021; Tsoukalas et al., 2024; Ren et al., 2025; Chen et al., 2025; Varambally et al., 2025; Liu et al., 2026). Much of this progress in Lean automation has focused on *autoformalization* and *automated theorem proving*. Autoformalization (Wu et al., 2022) concerns the translation of informal mathematical statements and arguments into formal statements or proofs. Automated theorem proving (Loveland, 2016), in contrast, focuses on proving an unsolved formal goal. This paper studies a later stage of the pipeline: *Once a Lean proof has passed verification, we ask whether it can be made shorter, more readable, and easier to maintain*.

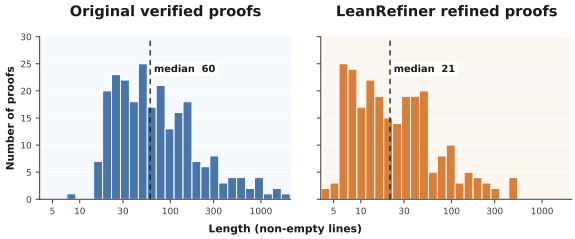

Figure 1: Proof length distributions on the combined main benchmark before and after LeanRefiner. Original proofs often contain redundancy that leads to unnecessary length, while LeanRefiner removes such redundancy without compromising correctness.

In practice, passing verification does not necessarily mean that a proof is readable and maintainable as code (Corbineau, 2007). Automatically generated Lean proofs often achieve correctness through verbose intermediate claims, repeated local reasoning, one-use auxiliary steps, or derivation fragments that could be consolidated. While these patterns are accepted by the verifier, they can reduce the long-term readability and maintainability of the proof. This observation motivates a natural post-processing problem that is separate from both theorem proving and autoformalization. In this paper, we define this task as *proof optimization after correctness*. Unlike theorem proving or autoformalization, it starts from an already verified Lean

proof rather than from an unsolved theorem or an informal mathematical text. The optimized proof must preserve the theorem statement and remain accepted by Lean. Crucially, it must also introduce no new proof holes, which in Lean are represented by the unfinished-proof placeholder `sorry`. Allowing new `sorry`s would hide missing proof obligations, so they are explicitly forbidden by the verifier gate (Moura and Ullrich, 2021). Figure 1 provides empirical evidence for this task: verified proofs often contain substantial redundancy, and this redundancy can be reduced while preserving correctness.

To address this problem, we propose **LeanRefiner**, a verifier-guided agentic system for proof optimization after correctness. LeanRefiner uses a staged two-agent design that combines theorem-level restructuring with local proof reduction. The *global restructuring agent* first uses a theorem-first global task, a semantic proof plan, and a compact external-lemma frontier to reconstruct a shorter proof route. Once the candidate is accepted by Lean, the system re-extracts its proof structure and passes the refreshed context to the *local reduction agent*, which searches for bounded replacements in redundant or compressible proof fragments. The two agents therefore use different Lean-derived views: global generation works from a staged theorem target and planning artifacts, while local reduction works from proof zones and verifier diagnostics in the accepted proof. However, they only propose candidate proofs. Final adoption is controlled by a *unified verifier-guided selector*, which accepts a candidate only if it passes Lean, preserves the theorem statement, introduces no new `sorry`, and reduces the current proof length. This design enables LeanRefiner to combine global proof reorganization with local proof compression while enforcing formal correctness as a hard constraint. Extensive experiments on multiple verified Lean proof collections demonstrate the effectiveness and robustness of LeanRefiner.

The main contributions are as follows:

- Proof optimization after correctness in Lean is formulated as verifier-constrained rewriting of already valid proofs, with hard gates for compilation, proof holes, and declaration preservation.
- We introduce LeanRefiner, an agentic system that combines theorem-level global restructuring with pattern-guided local reduction of bounded proof fragments.
- Extensive experiments on three verified Lean proof collections demonstrate that LeanRefiner can effectively reduce proof length while preserving formal correctness.

2 Related Work

Lean Theorem Proving. Lean theorem proving focuses on constructing proofs for formal goals within a proof assistant. Recent progress (Lu et al., 2025a; Ying et al., 2024; Lin et al., 2025; Song et al., 2024) in this area has been supported by scalable infrastructure, standardized benchmarks, and verifier-aware prover systems. LeanDojo (Yang et al., 2023) enables retrieval-augmented theorem proving through programmatic Lean interaction and repository-level proof-state extraction, while benchmarks such as MiniF2F and PutnamBench provide formalized competition-style problems for systematic evaluation (Zheng et al., 2021; Tsoukalas et al., 2024). Building on these resources, recent provers combine language-model generation with structured search and Lean feedback (Jiang et al., 2024; Lin et al., 2025; Wang et al., 2025). For example, DeepSeek-Prover-V2 (Ren et al., 2025) decomposes difficult theorems into subgoals for recursive proof generation, while Prover Agent (Baba et al., 2025) uses an agentic loop over informal reasoning, formal proof attempts, verifier feedback, and auxiliary lemma discovery. The Project Numina infrastructure also provides programmatic access to Lean through Lean's Language Server Protocol (LSP) with the Model Context Protocol (MCP) (Li et al., 2024; Dressler, 2025). This interface makes diagnostics, goal states, term-level information, and search tools available to external systems. These works primarily aim to prove formal goals that have not yet been solved. In contrast, LeanRefiner starts from a proof that already passes verification and studies how its structure can be shortened and reorganized after correctness is established (Gu et al., 2025; Zhou et al., 2024; Lu et al., 2026).

Autoformalization. Autoformalization (Lu et al., 2025b) aims to translate informal mathematical content into machine-checkable formal statements or proofs, thereby bridging natural-language mathematics and proof assistants. Recent work (Ren et al., 2025; Yin and Gao, 2025; Yang et al., 2023; Wang et al., 2024; Zhao et al., 2024) has improved this front-end translation pipeline through verification-aware training, semantic evaluation, and retrieval-based alignment between informal

and formal mathematics. For instance, Process-Driven Autoformalization uses Lean compiler feedback to train a process-supervised verifier for Lean 4 autoformalization (Lu et al., 2024), while ProofBridge retrieves semantically aligned formal examples through joint embeddings of natural-language and Lean theorem-proof pairs (Jana et al., 2025). This direction is complementary to LeanRefiner, since autoformalization concerns how formal content is obtained from informal mathematics, whereas LeanRefiner starts from an already verified Lean proof and optimizes its formal representation.

Proof Optimization and Proof Quality. Proof optimization (Rajaee et al., 2025; Gloeckle et al., 2025; George et al., 2025; Hubert et al., 2025; Zhang et al., 2025) is closest to the present work because it treats proof quality after verification as a research problem in its own right. ImProver (Ahuja et al., 2026) studies automated proof optimization in Lean with an inference-time agent for proof-state annotation, retrieval, and repair. ProofOptimizer (Gu et al., 2025) trains a learned proof simplifier through expert iteration and reinforcement learning, showing that long Lean proofs can often be compressed substantially. ProofJudge (SJCaldwell, 2026) studies proof quality through pairwise preference evaluation and emphasizes that human-facing proof quality is not determined by length alone. LeanRefiner also improves already verified proofs, but differs by using a verifier-guided global-to-local workflow that combines theorem-level restructuring with local reduction under explicit Lean hard gates.

3 Task Definition and Evaluation Protocol

We consider *proof optimization after correctness*, where the input is a Lean proof p that has already been verified without `sorry`, and the output is a candidate replacement proof p'. Unlike theorem proving, the goal is not to reconstruct a proof from the statement. Instead, the goal is to compress and restructure an existing proof by removing redundancy while preserving the original declarations. Before any length-based comparison, our evaluation protocol applies a hard verifier gate. A candidate proof is considered only if it passes Lean verification, contains no unfinished-proof placeholders, and preserves the declarations of the original proof. Declaration preservation is checked by comparing the normalized theorem or lemma declarations in the original and candidate proofs.

Evaluating proof quality after verification is not straightforward, since readability and style partly depend on mathematical and user preferences. Prior work has therefore considered richer objectives, such as user-defined metrics in ImProver (Ahuja et al., 2026) and Lean-specific token-count reduction in ProofOptimizer (Gu et al., 2025). In this paper, we use *proof-line reduction* as the main metric because it is transparent, reproducible, and less dependent on style conventions than weighted readability scores. For a proof p, let $\ell(p)$ denote the number of non-empty, non-comment lines in its proof body. For an original proof p and a candidate proof p', the relative **line reduction** is defined as

$$\mathrm{red}(p, p') = \frac{\ell(p) - \ell(p')}{\ell(p)}. \tag{1}$$

We report *pass rate* and *reduction rate* separately, so that verification success and proof shortening are not collapsed into a single score.

We also report three auxiliary diagnostic metrics: *token reduction*, *step reduction*, and *ProofJudge score*. Token reduction follows ProofOptimizer (Gu et al., 2025); step reduction estimates tactic-level action reduction from tactic heads and inline `by` fragments; and ProofJudge score uses ProofJudge (SJCaldwell, 2026) to assess clarity, maintainability, and style beyond raw length. These metrics are used only for analysis and are not directly optimized by LeanRefiner.

4 LeanRefiner

4.1 Overview

As shown in Figure 2, **LeanRefiner** is a verifier-guided two-agent system for *proof optimization after correctness*. Given a Lean proof whose target declaration already passes verification without `sorry`, LeanRefiner returns either a shorter verified replacement proof for the same declaration or the original proof when no valid shorter candidate is found. The key design principle is a strict separation between proposal and acceptance: *LLM agents propose candidate proof rewrites, while Lean verification determines which candidates are retained*.

The system consists of three components.

- **Global Restructuring Agent** proposes theorem-level replacement proofs using the target declaration, a proof-structure abstraction, and a compact lemma frontier.

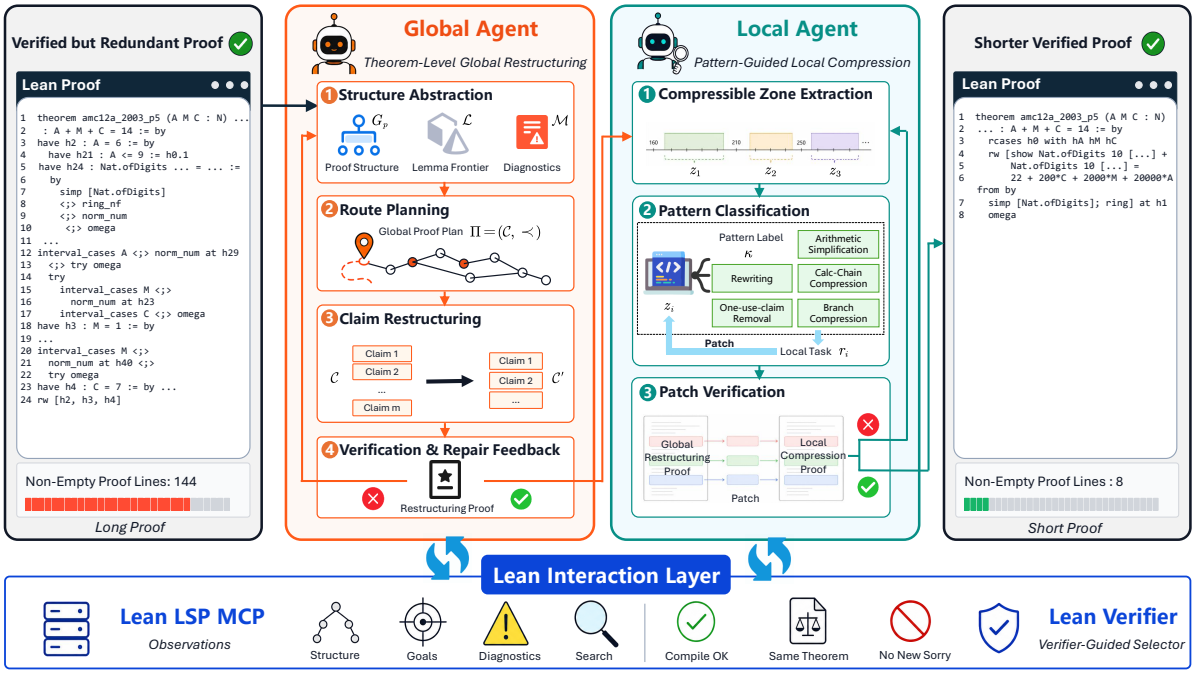

Figure 2: Overview of LeanRefiner. Starting from a verified but redundant proof, the system performs theorem-level global restructuring and pattern-guided local compression, while using Lean LSP MCP observations and verifier-guided selector to preserve correctness.

- **Local Reduction Agent** runs after an accepted global rewrite and proposes bounded patches around compressible regions in the refreshed proof.
- **Lean Interaction Layer** connects both agents to Lean observations and the Lean verifier through Lean LSP MCP. It enforces the verifier-guided selection rule and retains only candidates that satisfy the required validity.

This design is motivated by the multi-scale redundancy commonly found in generated Lean proofs. At the theorem level, redundancy may arise from unnecessary auxiliary claims, repeated intermediate derivations, or an overly indirect proof strategy, which calls for global rewriting. At the local level, redundancy often appears as repeated branch tails, verbose tactic blocks, or one-use helper claims, which are better handled through bounded patches. LeanRefiner therefore separates global proof reorganization from local compression, while applying the same verifier-guided acceptance rule to both stages through the Lean interaction layer. Implementation details are provided in Appendix C.

4.2 Verifier-Guided Optimization

The agents define how candidate proofs are proposed, but they do not decide which candidates are accepted. LeanRefiner treats acceptance as a verifier-guided optimization problem, where every candidate must satisfy the same formal constraints before its length is compared. Let s denote the normalized statement of the target declaration, and let p_0 denote the initial verified proof. LeanRefiner aims to find a shorter proof that remains valid under the same declaration. The idealized objective is therefore written as:

$$\begin{aligned} p^\star &= \arg\min_{p \in \mathcal{P}(s,p_0)} \ell(p) \\ \text{s.t.} \quad \mathsf{Valid}_{\mathsf{Lean}}(s,p) &= 1, \\ \mathsf{stmt}(p) &= s, \\ \mathsf{sorry}(p) &= 0. \end{aligned} \tag{2}$$

$\mathcal{P}(s, p_0)$ denotes the candidate proof space reachable from the initial proof p_0 for statement s. The predicate $\mathsf{Valid}_{\mathsf{Lean}}(s, p)$ requires Lean to accept the candidate proof for s. The constraint $\mathsf{stmt}(p) = s$ prevents the candidate from shortening the proof by changing the target declaration. Meanwhile, the constraint $\mathsf{sorry}(p) = 0$ rules out unfinished-proof placeholders such as `sorry`. As defined in Eq. (1), the length function $\ell(\cdot)$ is implemented as the number of non-empty, non-comment lines in the proof body.

LeanRefiner does not rely on the proposing agent to decide whether a generated proof can be accepted. Instead, acceptance is governed by a verifier-guided rule enforced by the Lean interac-

tion layer. Given a theorem statement s and a current verified proof p, the agent proposes a candidate proof q. The selector verifies q under the same formal acceptance conditions:

$$\begin{aligned} \mathsf{Valid}_{\mathsf{Lean}}(s,q) = 1, \quad & \mathsf{stmt}(q) = s, \\ \mathsf{sorry}(q) = 0, \quad & \ell(q) < \ell(p). \end{aligned} \tag{3}$$

If any condition fails, the candidate is rejected and the verifier diagnostics are returned to the agent as repair feedback. This verify-and-repair process is repeated until a valid shorter proof is found or the maximum number of rounds is reached. Let $\mathrm{Accept}(s,p,q)$ denote that q satisfies all validity and shortening conditions defined in Eq. (3). The selection rule is then

$$\mathrm{Select}_{\mathsf{Lean}}(s,p,q) = \begin{cases} q, & \text{if } \mathrm{Accept}(s,p,q), \\ p, & \text{otherwise.} \end{cases} \tag{4}$$

Thus, the current proof p is always retained as a verified fallback, and an agent proposal can update it only when the proposal is both valid and shorter. The same verifier-guided selection rule is applied to both global replacements and local patches.

4.3 Global Restructuring Agent

The global stage aims to generate a complete replacement proof for the target theorem. As shown in Figure 2, this stage treats proof shortening as a theorem level restructuring problem. It is designed for cases where the original Lean proof is already valid, but its *proof route* contains unnecessary intermediate claims or follows an overly indirect organization. Rather than editing redundant lines locally, the global restructuring agent first abstracts the verified proof into a compact structural representation, then plans a shorter global proof route and restructures the claims accordingly. The resulting candidate is then judged by Lean verification, which determines whether it can replace the original proof under the above verifier-guided selection rule.

Structure Abstraction The global agent begins with a deterministic abstraction of the input Lean proof. This abstraction extracts the prerequisite information needed for reconstruction while freezing the theorem context, including the imports, namespace context, declaration header, and normalized theorem statement. The global agent extracts these records by exploiting Lean-specific syntactic regularities and verifier diagnostics through programmable rules and lightweight heuristics. The resulting abstraction consists of three components: $\mathcal{L}$ records external lemmas and useful facts referenced by the original proof; G_p represents the concrete proof structure, including intermediate claims, proof blocks, and dependency relations; and $\mathcal{M}$ collects Lean diagnostic feedback from verification and previous failed attempts. Concrete examples and implementation details are provided in Appendix C.

Route Planning Based on G_p and $\mathcal{L}$, the agent first constructs a concise global proof plan, denoted by Π. Formally, Π is a structured summary of the original proof,

$$\Pi = (\mathcal{C}, \prec), \tag{5}$$

where $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ denotes the set of useful intermediate claims, and $\prec$ records the dependency order among these claims. m denotes the number of intermediate claims in the final plan, which may vary across different Lean proofs. The plan Π does not modify the Lean script directly. Instead, it guides the reconstruction toward a shorter theorem-level proof strategy while keeping the search grounded in the verified proof structure G_p and the lemma frontier $\mathcal{L}$.

Claim Restructuring After planning, the agent constructs a complete theorem-level proof for a fixed Lean target. The surrounding context, including imports, namespace information, and the declaration header, is kept unchanged, while the original proof body is replaced by a temporary `sorry` placeholder that must be discharged before acceptance. Using the plan Π, the lemma frontier $\mathcal{L}$, and the feedback set $\mathcal{M}$, the agent constructs a revised claim structure $\mathcal{C}'$ for the replacement proof. The set $\mathcal{C}'$ is expected to be more compact than $\mathcal{C}$, as auxiliary claims may be consolidated, reordered, simplified, or removed when they are unnecessary for the final derivation. During generation, the agent repeatedly queries the Lean interface for the current proof state, error messages, and relevant theorem-search results. These feedback signals help the agent revise its proof attempts according to the actual Lean environment rather than relying only on the generated proof plan. Finally, the output is a global candidate proof for the same theorem statement.

Verification and Repair Feedback Agent completion is not considered evidence of correctness. Each global candidate is inserted back into the complete Lean context and checked through the Lean

interaction layer. A candidate can replace the current proof only when it satisfies the verifier-guided selection rule in Eq. (4). Failed candidates produce diagnostics that are added to $\mathcal{M}$ as repair feedback for subsequent restructuring attempts. Once a valid shorter global candidate is accepted, it is passed to the local stage for finer-grained refinement.

4.4 Local Reduction Agent

The local stage performs pattern-guided compression on the proof accepted by the global stage. As shown in Figure 2, this stage focuses on bounded proof fragments rather than theorem-level restructuring. It is applied after global restructuring because the selected global proof usually provides a cleaner surface for identifying local redundancy. Such redundancy may appear as repeated case branches, verbose tactic sequences, or small helper blocks that can be safely simplified. In this way, the local agent avoids spending context on proof fragments that may already have disappeared during global restructuring.

Compressible Zone Extraction The local agent first scans the verified proof produced by the global stage and looks for local code patterns that are likely to be compressible. The extracted candidates are denoted by $\mathcal{Z}_p = \{z_i\}_{i=1}^{k}$. Each zone z_i records a contiguous region of the Lean proof that may be replaced. LeanRefiner also attaches nearby Lean warnings and diagnostics, since warnings often point to redundant tactics, unreachable code, or unnecessary simplification steps. Concrete examples of compressible zones are given in Appendix D.

Pattern Classification After extracting compressible zones, the local agent assigns each zone a coarse pattern label κ_i, yielding a local task $r_i(z_i, \kappa_i)$, where z_i is the candidate proof region and κ_i specifies the corresponding compression pattern. We use five coarse compression categories: arithmetic simplification, rewriting, `calc`-chain compression, one-use-claim removal, and branch compression. The label κ_i routes each local task to the corresponding specialized prompt. This decomposition keeps each LLM call focused and avoids packing all local regions into a single generic prompt, which may exceed the context window or obscure the proof information needed for a specific replacement. For each local task r_i, the local agent proposes a targeted patch that attempts to replace z_i with a shorter fragment suited to its compression pattern. The resulting patch is treated only as a candidate local replacement and is passed to the subsequent verification step before it can be incorporated into the proof.

Patch Verification A local patch is not accepted merely because it appears valid within the selected span. Instead, each proposed patch is spliced back into the complete Lean proof and checked through the Lean interaction layer. As specified by the verifier-guided selection rule in Eq. (4), the patch can replace the current span only when the resulting proof passes Lean verification, preserves the target declaration, contains no `sorry`, and reduces the proof length. If these conditions are not met, the patch is discarded and the current proof is retained. The diagnostics produced by failed patches are added to the local feedback context and can guide subsequent patch attempts, forming the verification feedback loop.

In summary, the two agents optimize the proof at complementary granularities. The global restructuring agent searches for a shorter theorem-level proof strategy for the fixed theorem statement, while the local reduction agent removes residual bounded redundancy from the verified proof. Both stages are mediated by the Lean interaction layer and governed by the same verifier-guided selection rule, ensuring that LeanRefiner shortens proofs without changing the proved statement or weakening formal correctness.

5 Experiments

5.1 Experimental Setup

Our experiments evaluate three questions: **(I)** whether LeanRefiner outperforms simpler alternatives, **(II)** whether its verifier-guided loop is robust to different proposal backends, and **(III)** how the global and local agents contribute to proof reduction.

The main benchmark contains 244 verified Lean proofs: 189 from `DeepSeek-Prover-V2`, 25 from `HILBERT`, and 30 from `SeedProver`. For `DeepSeek-Prover-V2` and `SeedProver`, we use the public Lean proofs released with the corresponding systems (Ren et al., 2025; Chen et al., 2025). For `HILBERT`, whose Lean proofs are not publicly released, we generate verified proofs by running the system (Varambally et al., 2025). These sources are used as input proofs rather than competing baselines. Detailed dataset statistics are provided in Appendix Tables 6 and 7. Controlled comparisons and ablations use the same fixed 85-

Table 1: Baseline comparison on the fixed 85-proof subset. Invalid outputs fall back to the original verified proof when computing reductions.

Method	Pass ↑	Line red. ↑	Tok. red. ↑
Symbolic	100.00%	3.47%	0.98%
Direct rewrite loop	40.00%	3.25%	2.95%
ImProver	89.41%	8.44%	5.94%
LeanRefiner	**100.00%**	**69.38%**	**70.33%**

proof repaired subset. Backend sensitivity is evaluated on a separate 24-task subset balanced across the three proof sources. The main benchmark reports proof-body line reduction, while other settings use whole-proof non-empty-line counts when proof bodies are not directly comparable; each result states its metric explicitly.

5.2 Comparison with Alternative Methods

We first examine whether LeanRefiner's gains can be explained by simpler alternatives. Table 1 reports controlled baseline results on the fixed repaired subset. Since the compared methods operate at different granularities, we report pass rate, strict whole-proof non-empty-line reduction, and an auxiliary Lean-token reduction under the same fallback rule. The symbolic cleanup and direct rewrite loop serve as internal rule-based controls, while the ImProver-style baseline adapts prior work on Lean proof optimization (Ahuja et al., 2026).

For the ImProver-style baseline, we use the released ImProver implementation with its length-optimization configuration, including Chain-of-States annotation, retrieval, few-shot examples, and iterative refinement (Ahuja et al., 2026). We adapt it to Mathlib v4.26.0 and evaluate it at the same proof level as the other baselines. Additional configuration details and compatibility adaptations are provided in Appendix E.1.

The results show that LeanRefiner's improvement cannot be explained by surface cleanup or direct rewriting alone. Symbolic cleanup preserves validity but yields only small reductions, while direct rewriting often fails the strict verifier gate. The ImProver-style baseline is stronger, but its aggregate reduction remains far below LeanRefiner under the same fallback protocol. The auxiliary token-reduction column leads to the same conclusion, even though token count is not directly optimized by LeanRefiner.

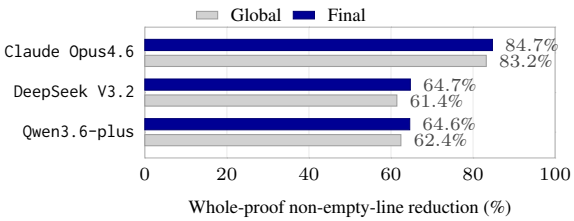

Figure 3: Backend sensitivity on a fixed 24-task subset. Bars report whole-proof non-empty-line reduction; darker bars denote final outputs.

5.3 Backend Sensitivity

Figure 3 reports a controlled backend-sensitivity experiment on 24 tasks balanced across the three proof sources. Only the proposal backend is varied, while the rest of the LeanRefiner pipeline is kept fixed. The metric is whole-proof non-empty-line reduction, consistent with the backend run summaries.

All three backends produce verified global outputs on all 24 tasks, suggesting that the verifier-guided optimization loop does not depend on a single proposal model. The reduction rate remains backend-sensitive: `Claude Opus 4.6` achieves the largest reduction on this subset, while `DeepSeek V3.2` and `Qwen 3.6-plus` give similar final reductions. We therefore use this experiment as a robustness check for LeanRefiner rather than as a general ranking of LLM backends for Lean proof optimization.

5.4 Detailed Main-Benchmark Analysis

We next examine where LeanRefiner's gains occur on the full 244-proof main benchmark. Figure 4 compares original and final proof-body lengths for each proof. Most points fall below the diagonal, indicating broad reductions rather than gains concentrated in a few outliers; the median proof length decreases from 60 to 20 lines.

Table 2 further breaks down the results by problem family. The input is the original verified proof from each proof source, and the final proof is the verifier-selected output after global restructuring and local reduction. Line reduction is the primary metric, while token reduction, step reduction, and ProofJudge score delta are reported only as auxiliary reference metrics.

The reductions are substantial across problem families, indicating that verified proofs still leave considerable room for optimization. The auxiliary token and proof-step metrics follow the same trend as line reduction, suggesting that the improvements

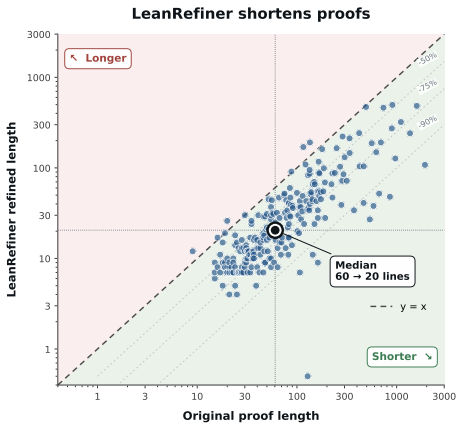

Figure 4: Per-proof proof-body length before and after LeanRefiner on the 244-proof main benchmark. Points below the dashed diagonal indicate shortened proofs; both axes use non-empty proof-body lines on a log scale.

Table 2: Main-benchmark results by problem family. Line reduction is the primary metric.

Family	n	Line red.	Tok. red.	Step red.	PJ Δ
Algebra	96	75.8%	77.8%	75.8%	+2.45
Number theory	78	69.5%	70.0%	68.6%	+1.90
Inequality	25	65.1%	53.1%	56.5%	+1.12
Arithmetic	21	83.4%	78.9%	84.0%	+2.76
Ind./seq.	18	48.9%	48.5%	48.3%	+1.94
Geometry	4	69.1%	77.9%	62.5%	+1.50
Analysis	2	71.0%	76.3%	69.7%	+0.50
Overall	**244**	**70.9%**	**68.5%**	**69.0%**	**+2.09**

are not merely formatting artifacts. ProofJudge also favors the final proofs on average, judging 203 of 244 pairs as improved, 21 as tied, and 20 as regressed, with a mean score delta of +2.09. We treat this judge-based result as supporting evidence rather than a substitute for Lean verification or the primary line-reduction metric.

Table 3 provides a descriptive source-family breakdown of the main benchmark. Table 4 evaluates the two agents on the same 85-proof repaired subset. The ablation shows a clear division of labor: local reduction alone is safe but modest, global restructuring accounts for most of the compression, and their composition achieves the best final reduction. This confirms that local reduction is not a replacement for global restructuring, but a complementary step that removes residual bounded redundancy after theorem-level reorganization.

We further examine local opportunities after global restructuring by scanning declaration-local verifier warnings. The warnings appear only in a subset of proofs and concentrate in a small number of local pattern families, supporting the use of targeted local compression rather than exhaustive local rewriting. Overall, LeanRefiner outperforms simpler baselines, remains effective across multiple proposal backends, and achieves broad reductions across proof families, with global restructuring providing the main compression and local reduction serving as a reliable final refinement step. Additional qualitative case studies and local-pattern analyses are provided in Appendix G.

Table 3: Reduction by proof source and problem family. Values are whole-proof non-empty-line reductions in percent; dashes denote missing source-family cells.

Source	Alg.	Num.	Ineq.	Arith.	Ind.	Geom.	Anal.	All
`DeepSeek`	65.9	68.1	47.4	79.6	57.5	35.9	57.8	68.3
`Hilbert`	79.5	79.8	76.5	–	–	75.5	–	79.3
`SeedProver`	78.3	64.1	56.8	–	45.3	–	70.9	60.8

Table 4: Two-agent ablation on the fixed subset.

Variant	Pass	Red.
Local only	100.00%	14.92%
Global only	100.00%	67.22%
Global + Local	**100.00%**	**69.38%**

6 Conclusion

We presented LeanRefiner, a verifier-guided two-agent system for optimizing already verified Lean proofs. LeanRefiner builds a shared proof-structure abstraction to coordinate theorem-level global restructuring, targeted local reduction, and verifier-guided selection. This design turns post-verification proof optimization into a structured optimization problem rather than a single-pass rewrite. Across three verified Lean proof collections, LeanRefiner consistently reduces proof length under strict verifier constraints. Controlled baselines, ablations, and backend-sensitivity experiments show that these gains come from the global-to-local agent design rather than surface cleanup or reliance on a single proposal backend. More broadly, our results suggest that proof assistants such as Lean can serve not only as final checkers for generated proofs, but also as feedback environments for agents that refine existing verified artifacts. We view LeanRefiner as a step toward verifier-grounded proof engineering systems, where LLM agents use proof-assistant feedback not only to construct proofs, but also to improve the structure of existing verified artifacts.

Limitations

LeanRefiner refines already verified Lean proofs rather than proving new theorems from scratch. This setting enables optimization under hard verifier gates, but the current proposal-based procedure does not guarantee globally shortest proofs, especially when improvements require new auxiliary lemmas, alternative mathematical arguments, or deeper library discovery. Moreover, our evaluation primarily uses non-empty proof-body line reduction. This metric is reproducible and captures an important form of redundancy, but it does not fully measure readability, maintainability, pedagogical clarity, or robustness under library changes. Future work could incorporate richer quality objectives, such as readability-aware compression, proof robustness analysis, and preference judgments from Lean users. In addition, LeanRefiner also incurs computational overhead because it uses multi-round candidate generation and repeated Lean verification. Improving candidate ranking, caching verifier results, and designing adaptive verification schedules could make verifier-guided proof refinement more efficient. Finally, proof compression is not a complete proxy for mathematical understanding, but a step toward proof engineering systems that use proof-assistant feedback to improve existing verified artifacts.

References

Riyaz Ahuja, Jeremy Avigad, Prasad Tetali, and Sean Welleck. 2026. Improver: Agent-based automated proof optimization. *Preprint*, arXiv:2410.04753.

Kaito Baba, Chaoran Liu, Shuhei Kurita, and Akiyoshi Sannai. 2025. Prover agent: An agent-based framework for formal mathematical proofs. *arXiv preprint arXiv:2506.19923*.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, and 17 others. 2025. Seed-prover: Deep and broad reasoning for automated theorem proving. *Preprint*, arXiv:2507.23726.

Pierre Corbineau. 2007. A declarative language for the coq proof assistant. In *International Workshop on Types for Proofs and Programs*, pages 69–84. Springer.

Oliver Dressler. 2025. Lean LSP MCP: Tools for agentic interaction with the Lean theorem prover.

Robert Joseph George, Suozhi Huang, Peiyang Song, and Anima Anandkumar. 2025. Leanprogress: Guiding search for neural theorem proving via proof progress prediction. *arXiv preprint arXiv:2502.17925*.

Fabian Gloeckle, Alex Gu, Gabriel Synnaeve, and Amaury Hayat. 2025. Reinforcement learning for hierarchical proof generation in lean 4. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Alex Gu, Bartosz Piotrowski, Fabian Gloeckle, Kaiyu Yang, and Aram H. Markosyan. 2025. Proofoptimizer: Training language models to simplify proofs without human demonstrations. *arXiv preprint arXiv:2510.15700*.

John Harrison. 1996. Formalized mathematics. Technical Report 36, Turku Centre for Computer Science.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, and 1 others. 2025. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pages 1–3.

Prithwish Jana, Kaan Kale, Ahmet Ege Tanriverdi, Cruise Song, Sriram Vishwanath, and Vijay Ganesh. 2025. Proofbridge: Auto-formalization of natural language proofs in lean via joint embeddings. *arXiv preprint arXiv:2510.15681*.

Dongwei Jiang, Marcio Fonseca, and Shay B Cohen. 2024. Leanreasoner: Boosting complex logical reasoning with lean. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7497–7510.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, and 1 others. 2024. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, and 1 others. 2025. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*.

Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, Lihong Zhi, Jia Li, and Wenda Li. 2026. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. *Preprint*, arXiv:2601.14027.

Donald W Loveland. 2016. *Automated theorem proving: A logical basis*. Elsevier.

Jialin Lu, Kye Emond, Kaiyu Yang, Swarat Chaudhuri, Weiran Sun, and Wuyang Chen. 2025a. Lean finder: Semantic search for mathlib that understands user intents. *arXiv preprint arXiv:2510.15940*.

Jialin Lu, Soonho Kong, Rodrigo Stehling, Kaiyu Yang, Zhangyang Wang, Weiran Sun, and Wuyang Chen. 2026. Lean refactor: Multi-objective controllable proof optimization via agentic strategy search. *arXiv preprint arXiv:2605.20244*.

Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. 2025b. Formalalign: Automated alignment evaluation for autoformalization. In *International Conference on Learning Representations*, volume 2025, pages 102122–102149.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, and 1 others. 2024. Process-driven autoformalization in lean 4. *arXiv preprint arXiv:2406.01940*.

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

Sara Rajaee, Kumar Pratik, Gabriele Cesa, and Arash Behboodi. 2025. Local look-ahead guidance via verifier-in-the-loop for automated theorem proving. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16023–16040.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

SJCaldwell. 2026. Proofjudge. Hugging Face dataset. Version 0.1.0; accessed 2026-05-26.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Lean copilot: Large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*.

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

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. 2025. Hilbert: Recursively building formal proofs with informal reasoning. *ArXiv*, abs/2509.22819.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, and 1 others. 2025. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, and 1 others. 2024. Lego-prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, volume 2024, pages 31566–31597.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus N. Rabe, Charles E. Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Preprint*, arXiv:2205.12615.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612.

David Yin and Jing Gao. 2025. Generating millions of lean theorems with proofs by exploring state transition graphs. *arXiv preprint arXiv:2503.04772*.

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

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. 2025. Leanabell-prover: Posttraining scaling in formal reasoning. *arXiv preprint arXiv:2504.06122*.

Xueliang Zhao, Lin Zheng, Haige Bo, Changran Hu, Urmish Thakker, and Lingpeng Kong. 2024. Subgoalxl: Subgoal-based expert learning for theorem proving. *arXiv preprint arXiv:2408.11172*.

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

Jin Zhou, Yuhuai Wu, Qiyang Li, and Roger Grosse. 2024. Refactor: Learning to extract theorems from proofs. In *International Conference on Learning Representations*, volume 2024, pages 40601–40621.

Appendix

A Notation

Table 5 summarizes the notation used in the main text. The notation is intended to distinguish the verified input proof, the proof body being optimized,

and the Lean-derived structural signals used by the global and local agents.

B LeanRefiner Pseudocode

Algorithm 1 gives the high-level workflow. The procedure separates proposal from acceptance: global restructuring and local reduction can propose many candidates, but a candidate is accepted only after the same verifier hard gate and length-based selection rule used in the main text.

C Implementation Artifacts for Proof Reconstruction

This appendix explains how the symbolic objects used in the method section are materialized in the implementation. We focus on three runtime artifacts: the proof-structure summary G_p, the lemma frontier $\mathcal{L}$, and the verifier-feedback summary $\mathcal{M}$. Here p denotes the current Lean proof being optimized. The subscript in G_p means that the structure is extracted from proof p.

G_p is not a full semantic proof graph. It is a compact, operational summary of the current Lean proof. It records which declaration is being optimized, which parts of the file must remain frozen, where the proof body begins and ends, which local claims or proof blocks appear, and which blocks appear structurally repetitive. Its purpose is to help the global agent see the shape of the existing proof without requiring it to reason over the entire raw file at once.

$\mathcal{L}$ is a compact lemma frontier rather than an exhaustive library search. It is built from theorem names, facts, and tactic-level cues already present in the original proof or strongly suggested by the current proof context. The role of $\mathcal{L}$ is to keep the rewrite grounded in the source proof and to avoid losing useful library hints during reconstruction.

$\mathcal{M}$ summarizes Lean feedback in a form that the agent can use during repair. It records whether the file compiles, whether the target statement is unchanged, whether any sorry remains, and what errors or warnings Lean reports. When a rewrite fails, $\mathcal{M}$ helps distinguish different failure modes, such as unresolved identifiers, tactic failures, remaining goals, statement changes, or newly introduced proof holes. The agent may use this feedback to revise its proof, but $\mathcal{M}$ itself is not a scoring metric; final acceptance still depends on Lean verification.

For example, consider `amc12a_2003_p5`. The original proof separately proves $A = 6$, $M = 1$, and $C = 7$, and each part repeatedly expands the two `Nat.ofDigits` expressions before closing the resulting arithmetic goal. For this proof, G_p is the proof-structure abstraction extracted from the current Lean proof. It contains two complementary views. The first is a structural dependency graph whose nodes correspond to major intermediate claims and proof blocks, such as the digit-expansion step, the derived arithmetic relation, and the three subclaims for A, M, and C. Edges record which later claims depend on which earlier facts. The second is a concise natural-language proof-structure summary that describes the same route at a higher level: expand the two digit expressions, derive the arithmetic constraint $C + 10M + 100A = 617$, and use the digit bounds to conclude $A + M + C = 14$. Thus, G_p is not a collection of low-level syntax fields. It is a compact representation of the proof route and its dependencies, which allows the global agent to see that the repeated subproofs can be replaced by deriving the central digit equation once and closing the final goal arithmetically.

In the same example, $\mathcal{L}$ contains reusable cues already visible in the source proof, including the use of `Nat.ofDigits` for digit expansion and arithmetic tactics such as `ring` and `omega`. This frontier is intentionally compact: it is not a full Mathlib retrieval result, but a small set of facts and tactics that keeps the rewrite grounded in the original proof.

The verifier-feedback summary $\mathcal{M}$ records Lean's response to the current proof and to later rewrite attempts. For the original proof, $\mathcal{M}$ records that the file compiles and that the target declaration is unchanged. For a failed rewrite, it records compact feedback such as an unresolved identifier, an unfinished goal, a changed declaration, a tactic failure, or a remaining `sorry`. The agent may use this feedback during repair, but final acceptance still depends on the Lean hard gate.

C.1 Solve-Stage Prompt

LeanRefiner uses a bundle-driven solve prompt. At runtime, the placeholder `{{SYSTEM_PROMPT}}` is expanded with the global system prompt, and the path placeholders are filled with the concrete per-instance bundle paths.

LeanRefiner solve-stage prompt

```
You are LeanRefiner GLOBAL_GEN.

Role:
```

Table 5: Notation used in the task definition, system description, and evaluation protocol.

Symbol	Meaning
F, F'	Original verified Lean proof and candidate replacement proof.
s	Target theorem or declaration statement.
p, p'	Original proof body and candidate replacement proof body for the same statement.
$p_0, p^\star, q$	Initial verified proof, ideal optimized proof, and proposed candidate proof.
p_t, p_t^{G}	Current accepted proof and verifier-selected global proof at optimization round t, used in the algorithmic description.
$\ell(p)$	Length of proof body p, measured by non-empty, non-comment proof-body lines in the main benchmark.
$\mathcal{P}(s, p_0)$	Candidate proof space reachable from the initial proof p_0 for statement s.
$\mathsf{Valid}_{\mathsf{Lean}}(s, p)$	Lean verifier predicate indicating that proof p proves statement s under the current environment.
$\mathsf{stmt}(p)$	Normalized statement or declaration signature preserved by the hard gate.
$\mathsf{sorry}(p)$	Number of unfinished-proof placeholders in proof p.
$\mathrm{Accept}(s, p, q)$	Predicate indicating that candidate q passes the hard verifier gate and is shorter than current proof p.
$\mathrm{Select}_{\mathsf{Lean}}(s, p, q)$	Verifier-guided selector that returns q when it is accepted and otherwise retains p.
G_p	Lightweight blueprint of proof/file structure used by the global agent, including declaration spans, statement records, proof statistics, and dependency cues.
$\mathcal{L}$	Lemma frontier supplied to the proposal agents.
$\mathcal{M}$	Lean diagnostics and verifier messages available.
$\Pi = (\mathcal{C}, \prec)$	Global proof plan consisting of useful intermediate claims and their dependency order.
$\mathcal{C}, \mathcal{C}'$	Original planned claim set and revised compact claim structure.
c_i, m	An intermediate claim in the global plan and the number of such claims.
$\mathcal{Z}_p = \{z_i\}_{i=1}^k$	Compressible zones extracted from proof p.
κ_i	Coarse compression-pattern label assigned to zone z_i.
$r_i(z_i, \kappa_i)$	Local reduction task built from a zone and its pattern label.
U	A bounded local proof unit considered by the local agent.
$\mathrm{red}(p, p')$	Relative line reduction $(\ell(p) - \ell(p'))/\ell(p)$.

```
- You are the generation stage inside the LeanRefiner
      orchestrator.
- Your job is to improve an existing Lean proof file while
      preserving correctness.

Global hard constraints:
- Never change theorem or lemma statements.
- Treat theorem/lemma headers as immutable. If any attempt
      changes a statement, revert that header exactly and
      continue from the last valid version.
- Never introduce new `sorry`, `admit`, `axiom`, or
      placeholders.
- Prefer the smallest valid edit set that improves the
      current proof.
- Stay close to the existing structure unless the task
      instructions explicitly call for a rewrite.
- Output must follow the task-specific format exactly.

Priority order:
1. Preserve statement semantics.
2. Eliminate proof holes and compile errors.
3. Reduce proof size and redundancy.
4. Improve structure and readability.

Stage:
- LEANREFINER_SOLVE

Role:
- You are the solve-stage entry prompt for LeanRefiner v3
      .5.
- The solve-stage setup is intentionally minimal.
- Do not invent a separate planning stack here.
- Read the prepared minimal solve bundle and solve the
      target by editing only `target.lean`.

Solve-facing bundle:
- `target.lean`
  - the only file you should edit
  - contains the main theorem statement and an initial `
      sorry`
- `nl_proof.md`
  - concise natural-language proof route for the main
      theorem
  - use it as strategic guidance, not as a literal script
- `external_lemmas.json`
  - L: compact heuristic frontier of likely useful external
       theorems
- `lean_diagnostics.json`
  - M: lightweight Lean diagnostic feedback for the
      original target context
- `bundle_manifest.md`
  - artifact guide

Non-negotiable rules:
- Edit only `target.lean`.
- Do not create, edit, delete, or reinitialize any other
      file in `solve_bundle/`.
- In particular, never create or modify `.lake/`, `lean-
      toolchain`, or `lake-manifest.json`.
- Never change the theorem statement.
- Never introduce `axiom`.
- Never leave `sorry`, `admit`, or placeholders in the
      final `target.lean`.
- Prefer solving the main theorem directly.
- Introduce helper lemmas only when they materially
      simplify or unblock the main theorem.
- Prefer semantic lemma names over generated names.
- Do not use generated helper names such as `round1_*`, `
      round2_*`, `tmp_*`, `helper1`, or similar sequence-
      based labels.
- If you introduce a helper lemma, give it a short
      meaningful name that reflects the mathematical fact
      or proof role.

Mode start sequence:
1. Read `bundle_manifest.md`.
2. Read `target.lean`.
3. Read `nl_proof.md`.
4. Read `external_lemmas.json` when theorem search or
      premise selection is needed.
5. Read `lean_diagnostics.json` for Lean diagnostic
      feedback when available.
6. Follow the selected mode doctrine.

Primary objective:
```

Algorithm 1 LeanRefiner verifier-guided proof optimization

Require: Verified Lean proof F, target statement s, initial proof p_0, maximum rounds T
Ensure: Verified proof p_T or the best verified fallback proof

```
Check that F compiles and extract the root statement s
Set current proof p ← p_0
Extract declaration spans, proof statistics, diagnostics M, and lemma frontier L
for t = 1 to T do
    Build the staged target proof and natural-language proof plan for the global agent
    Build the global prompt from the staged target, plan, L, and M
    Call the global LLM backend to produce a proof snapshot
    Verify the snapshot with Lean and keep it only if it satisfies the hard verifier gate
    if a verified global candidate is shorter than p then
        Update p to the shortest verified global candidate
        Re-extract declaration spans, M, and local proof zones from the updated proof
    end if
    Construct local triggers from proof zones and verifier diagnostics
    for each local trigger do
        Identify a bounded local unit U with an editable line range and immutable context
        Try deterministic warning-guided cleanup for U
        If no deterministic rewrite applies, call the local LLM backend for a bounded patch
        Splice each proposed replacement into the whole proof
        Verify each spliced proof with Lean
        if a verified local replacement shortens the whole proof then
            Accept the shortest verified replacement and update p
        end if
    end for
    Retain p as fallback when no valid shorter candidate exists
end for
return p
```

```
- Turn `target.lean` into a hole-free, compile-targeting
      proof of the main theorem.

Secondary objective:
- Keep the top-level proof shape short.
- Avoid unnecessary top-level lemma proliferation.

Finish policy:
- Stop only when `target.lean` is sorry-free and compiles
      without errors.
- Before ending with `END_REASON:COMPLETE`, run `
      lean_diagnostic_messages` one more time on the whole
      file.
- If `lean_diagnostic_messages` still reports any error
      diagnostics, or if there are still `sorry`s/errors,
      do not emit `COMPLETE`.
- If you believe the environment is missing Lean project
      metadata or `lake`, do not try to repair the
      environment by writing `.lake/`, `lean-toolchain`, or
       `lake-manifest.json` inside `solve_bundle/`;
      continue working only by editing `target.lean`.
- When finished, output exactly one line:
END_REASON:COMPLETE

Runtime context:

Bundle directory:
{{BUNDLE_DIR}}

Bundle manifest:
{{BUNDLE_MANIFEST_PATH}}

Working file:
{{TARGET_FILE}}

Natural-language proof path:
{{NL_PROOF_PATH}}

External lemma frontier path:
{{EXTERNAL_LEMMAS_PATH}}

Lean diagnostic feedback path:
{{LEAN_DIAGNOSTICS_PATH}}

Preclassified mode:
{{PRESELECTED_MODE}}

Mode metrics:
{{MODE_METRICS}}
```

D Local Reduction Task Examples

LeanRefiner's local stage extracts small editable zones from the verified proof and assigns each zone a coarse reduction-family hint κ_i. These hints are used only to choose the local prompt template and task priority; every candidate replacement is still accepted only after full Lean verification.

Arithmetic compression. A zone is assigned an arithmetic hint when it contains a short arithmetic cleanup pattern, such as `field_simp` followed by `linarith` or `nlinarith`. The local agent attempts to replace such a span with a shorter arithmetic proof while preserving the surrounding Lean context.

Arithmetic compression

```
-- before
have h1 : x ≠ 0 := by
  positivity
field_simp [h1]
linarith

-- after
field_simp
linarith
```

Rewrite compression. A zone is assigned a rewrite hint when the local proof is dominated by repeated `rw`, `simp`, or `simpa` steps. The local agent attempts to merge adjacent rewrites or replace a chain of local rewrites with a shorter simplification step.

Rewrite compression

```
-- before
rw [h1]
rw [h2]
simpa using h3

-- after
simpa [h1, h2] using h3
```

Calc-chain compression. A zone is assigned a calc-chain hint when the editable span is a short `calc` block whose intermediate steps are redundant or can be fused. The local agent attempts to shorten the chain without changing the endpoints.

Calc-chain compression

```
-- before
calc
  a = b := by rw [h1]
  _ = c := by rw [h2]
  _ = d := by simpa using h3

-- after
calc
  a = c := by rw [h1, h2]
  _ = d := by simpa using h3
```

One-use claim elimination. A zone may contain an intermediate `have` claim that is used only once immediately after it is introduced. In this case, the local agent attempts to inline the claim or replace the producer-consumer pair by a shorter direct proof.

One-use claim elimination

```
-- before
have h : P := by
  exact hp
exact h

-- after
exact hp
```

Branch compression. A zone is assigned a branch-compression hint when several sibling proof branches have repeated endings, such as repeated contradictions or repeated arithmetic closings. The local agent edits the whole sibling branch block rather than a single branch in isolation.

Branch compression

```
-- before
cases h with
| inl h1 =>
    exfalso
    linarith
| inr h2 =>
    exfalso
    linarith

-- after
cases h with
| inl h1 => linarith
| inr h2 => linarith
```

These examples are schematic. In the implementation, each proposed replacement is inserted back into the original Lean file and accepted only if the resulting proof compiles and has a shorter proof body.

E Datasets and Metrics

Table 6 summarizes the data sources and the metric used in each experimental setting. The main benchmark reports proof-body line reduction because it evaluates the object LeanRefiner directly optimizes. For the main benchmark, we additionally report auxiliary Lean-token, proof-step, and ProofJudge metrics as post-hoc reference signals rather than optimization objectives. The controlled baseline table additionally reports an auxiliary Lean-token metric under the same hard-gate fallback, while the ablation and backend-sensitivity experiments use whole-proof non-empty-line metrics where proof-body boundaries are not directly comparable across systems.

E.1 ImProver Baseline Configuration

We run the released ImProver length-optimization method and keep the configuration used in the ImProver paper. The method uses Chain-of-States

Table 6: Dataset and metric summary. Whole-proof metrics are used only when method outputs do not share a stable proof-body boundary.

Setting	Proofs	Sources	Problem families	Metric
Main full benchmark	244	`DeepSeek`, `HILBERT`, `SeedProver`	AIME, AMC, MathD, IMO-style problems, induction and number-theory instances	Primary: proof-body line reduction; auxiliary: Lean-token, proof-step, and ProofJudge metrics
Raw source pool	272	Full processed raw pools for the three sources	Algebra, number theory, inequality, arithmetic, induction/sequence, geometry, and analysis categories	Whole-proof line and non-empty-line statistics for dataset characterization
Baseline comparison	85	Repaired compile-clean subset with 30 DeepSeek, 25 HILBERT, and 30 SeedProver proofs	Same-input subset used by LeanRefiner and baselines	Whole-proof non-empty-line reduction; auxiliary Lean-token metric
Two-agent ablation	85	Same repaired subset as the baseline comparison	Same problem proofs for local-agent-only, global-agent-only, and composed two-agent variants	Whole-proof non-empty-line reduction
Backend sensitivity	24	Balanced subset with 8 proofs from each proof source	Fixed tasks, prompt, verifier gate, and selection rule across proposal backends	Whole-proof non-empty-line reduction

Table 7: Automatic problem-family counts over 272 raw source proofs. The manual-review column marks cases whose automatic category should be checked before using the label in a strong claim.

Problem family	DeepSeek	HILBERT	SeedProver	Total	Manual review
Algebra	82	14	2	98	1
Number theory	73	9	10	92	11
Inequality	22	1	9	32	29
Arithmetic	24	0	0	24	24
Induction or sequences	12	0	8	20	11
Geometry	3	1	0	4	3
Analysis	1	0	1	2	2

annotation, retrieval over syntax and Mathlib documents, 10 retrieved metric examples, 5 refinement iterations, best-of-3 sampling at each iteration, and keep-best selection. Our changes are limited to compatibility and evaluation setup. We adapt the vendored code to Mathlib v4.26.0, evaluate the final declaration in each repaired input file to match our file-level benchmark, and increase the structured-output budget and runtime guards to avoid truncation artifacts.

The original ImProver paper reports GPT-4o as the base generator. In our comparison, ImProver is run with DeepSeek V4 Pro, which is a stronger backend than the DeepSeek V3.2 model used in our main LeanRefiner experiments. These changes do not alter ImProver's prompt, refinement strategy, or length metric.

F Problem Families and Source Examples

Table 7 reports an automatic problem-family classification over the raw Lean proofs. The labels are generated from proof names, formal statement comments, declaration signatures, and lightweight Lean keyword cues. They are intended for dataset characterization and appendix-level analysis rather than for strong claims about mathematical difficulty.

G Qualitative Cases and Boundary Examples

Figure 5 illustrates one representative DeepSeek before/after case. A second HILBERT case, `imo2020_p2_algebra`, is reproduced as an extreme successful compression example. Additional local-reduction and boundary examples are dis-

Table 8: Declaration-local warning scan over 232 post-restructuring candidates.

Warning family	Count	Proofs	Implication for local reduction
`unused_tactic`	250	14	Dead tactic tails and redundant-step cleanup have clear locality, making them suitable as warning-first priority tasks
`unreachable_tactic`	236	11	Branches or tactic tails often contain directly deletable unreachable suffixes, making them suitable for zone-first closure
`unnecessary_seq_focus`	111	10	Compressible sequencing patterns exist in multi-branch or local tactic blocks
`unused_simp_arg`	50	8	Suitable for a safe cleanup path with deterministic-first handling and proposal fallback
`unnecessary_simpa`	6	6	Less frequent, but a high-confidence, low-risk local reduction opportunity

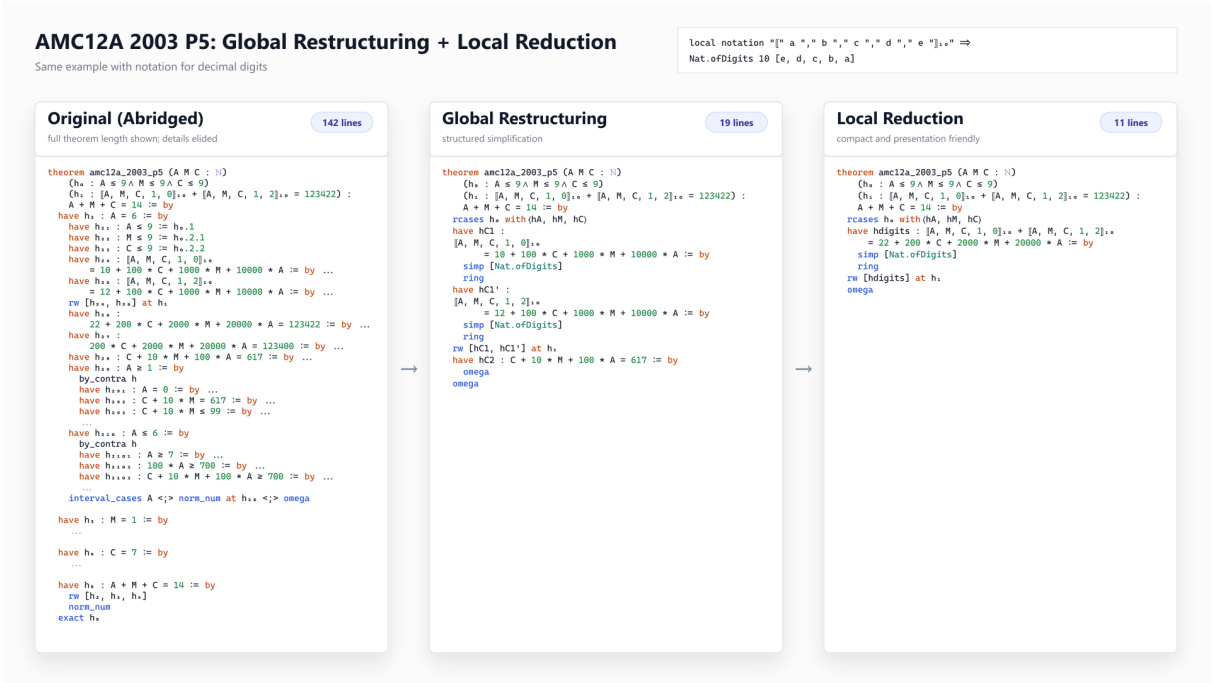

Figure 5: A representative AMC12A 2003 P5 case showing proof reduction from the original proof to global restructuring and then local reduction.

cussed through the cases below. The main text reports only aggregate conclusions; concrete before/after proof fragments and run logs will continue to be supplemented in the appendix.

H Additional Distribution Analysis

Figure 1 gives the proof-level distribution of whole-proof non-empty lines on the combined main benchmark. This distributional view complements the aggregate reduction table: LeanRefiner not only reduces the total line count, but also shifts many proofs toward shorter verified Lean programs.

I Additional Local Warning Analysis

Table 8 reports the declaration-local warning-family breakdown referenced in the main text. These warnings are used as diagnostic anchors for local reduction, not as standalone optimization objectives.

The corresponding Lean proofs are reproduced below. Listing A shows the original verified input, Listing B shows the verifier-selected global restructuring output, and Listing C shows the final proof after local reduction.

Listings D–F show an additional HILBERT algebra case. Here, a helper-heavy verified input is reduced from 538 to 27 whole-proof non-empty lines, a 94.98% reduction; the final verifier-selected proof is the same as the accepted global restructuring output because no shorter local patch was accepted.

Listing A: original verified input for `amc12a_2003_p5.lean`.

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- The sum of the two 5-digit numbers $AMC10$ and $AMC12$ is $123422$. What is $A+M+C$?

$ \mathrm{(A) \ } 10\qquad \mathrm{(B) \ } 11\qquad \mathrm{(C) \ } 12\qquad \mathrm{(D) \ } 13\qquad \mathrm{(E) \ } 14 $ Show that it is
↪  \mathrm{(E)}\ 14.-/
theorem amc12a_2003_p5 (A M C : ℕ) (h₀ : A ≤ 9 ∧ M ≤ 9 ∧ C ≤ 9)
    (h₁ : Nat.ofDigits 10 [0, 1, C, M, A] + Nat.ofDigits 10 [2, 1, C, M, A] = 123422) :
    A + M + C = 14 := by
  have h₂ : A = 6 := by
    have h₂₁ : A ≤ 9 := h₀.1
    have h₂₂ : M ≤ 9 := h₀.2.1
    have h₂₃ : C ≤ 9 := h₀.2.2
    have h₂₄ : Nat.ofDigits 10 [0, 1, C, M, A] = 10 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    have h₂₅ : Nat.ofDigits 10 [2, 1, C, M, A] = 12 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    rw [h₂₄, h₂₅] at h₁
    have h₂₆ : 10 + 100 * C + 1000 * M + 10000 * A + (12 + 100 * C + 1000 * M + 10000 * A) = 123422 := by
      linarith
    have h₂₇ : 22 + 200 * C + 2000 * M + 20000 * A = 123422 := by
      linarith
    have h₂₈ : 200 * C + 2000 * M + 20000 * A = 123400 := by
      omega
    have h₂₉ : C + 10 * M + 100 * A = 617 := by
      omega
    have h₃₀ : A ≥ 1 := by
      by_contra h
      push_neg at h
      have h₃₁ : A = 0 := by
        omega
      rw [h₃₁] at h₂₉
      have h₃₂ : C + 10 * M = 617 := by
        omega
      have h₃₃ : C + 10 * M ≤ 99 := by
        have h₃₄ : M ≤ 9 := h₀.2.1
        have h₃₅ : C ≤ 9 := h₀.2.2
        omega
      omega
    have h₃₁ : A ≤ 6 := by
      by_contra h
      push_neg at h
      have h₃₂ : A ≥ 7 := by omega
      have h₃₃ : 100 * A ≥ 700 := by
        omega
      have h₃₄ : C + 10 * M + 100 * A ≥ 700 := by
        omega
      have h₃₅ : C + 10 * M + 100 * A = 617 := by
        omega
      omega
    interval_cases A <;> norm_num at h₂₉ ⊢ <;>
      (try omega) <;>
      (try
        {
          have h₃₆ : M ≤ 9 := h₀.2.1
          have h₃₇ : C ≤ 9 := h₀.2.2
          interval_cases M <;> norm_num at h₂₉ ⊢ <;>
            (try omega) <;>
            (try
              {
                interval_cases C <;> norm_num at h₂₉ ⊢ <;>
                  omega
              })
        })
    <;> omega

  have h₃ : M = 1 := by
    have h₃₁ : A ≤ 9 := h₀.1
    have h₃₂ : M ≤ 9 := h₀.2.1
    have h₃₃ : C ≤ 9 := h₀.2.2
    have h₃₄ : Nat.ofDigits 10 [0, 1, C, M, A] = 10 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    have h₃₅ : Nat.ofDigits 10 [2, 1, C, M, A] = 12 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    rw [h₃₄, h₃₅] at h₁
    have h₃₆ : 10 + 100 * C + 1000 * M + 10000 * A + (12 + 100 * C + 1000 * M + 10000 * A) = 123422 := by
      linarith
    have h₃₇ : 22 + 200 * C + 2000 * M + 20000 * A = 123422 := by
      linarith
    have h₃₈ : 200 * C + 2000 * M + 20000 * A = 123400 := by
      omega
    have h₃₉ : C + 10 * M + 100 * A = 617 := by
      omega
    rw [h₂] at h₃₉
    have h₄₀ : C + 10 * M = 17 := by
      omega
    have h₄₁ : M ≤ 1 := by
      by_contra h
      push_neg at h
      have h₄₂ : M ≥ 2 := by omega
      have h₄₃ : C + 10 * M ≥ 20 := by
        have h₄₄ : C ≥ 0 := by omega
        nlinarith
      omega
    interval_cases M <;> norm_num at h₄₀ ⊢ <;>
      (try omega) <;>
      (try
        {
          have h₄₂ : C ≤ 9 := h₀.2.2
          interval_cases C <;> norm_num at h₄₀ ⊢ <;>
            (try omega)
        })
    <;> omega

  have h₄ : C = 7 := by
```

```
    have h41 : A ≤ 9 := h0.1
    have h42 : M ≤ 9 := h0.2.1
    have h43 : C ≤ 9 := h0.2.2
    have h44 : Nat.ofDigits 10 [0, 1, C, M, A] = 10 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    have h45 : Nat.ofDigits 10 [2, 1, C, M, A] = 12 + 100 * C + 1000 * M + 10000 * A := by
      simp [Nat.ofDigits, List.cons, List.append, List.map, List.foldr, Nat.mul_add, Nat.add_mul]
      <;> ring_nf
      <;> norm_num
      <;> omega
    rw [h44, h45] at h1
    have h46 : 10 + 100 * C + 1000 * M + 10000 * A + (12 + 100 * C + 1000 * M + 10000 * A) = 123422 := by
      linarith
    have h47 : 22 + 200 * C + 2000 * M + 20000 * A = 123422 := by
      linarith
    have h48 : 200 * C + 2000 * M + 20000 * A = 123400 := by
      omega
    have h49 : C + 10 * M + 100 * A = 617 := by
      omega
    rw [h2, h3] at h49
    norm_num at h49 ⊢
    <;> omega

  have h5 : A + M + C = 14 := by
    rw [h2, h3, h4]
    <;> norm_num

  apply h5
```

Listing B: global restructuring output for `amc12a_2003_p5.lean`.

```
import Mathlib


open BigOperators Real Nat Topology Rat

/-- The sum of the two 5-digit numbers $AMC10$ and $AMC12$ is $123422$. What is $A+M+C$?

$ \mathrm{(A) \ } 10\qquad \mathrm{(B) \ } 11\qquad \mathrm{(C) \ } 12\qquad \mathrm{(D) \ } 13\qquad \mathrm{(E) \ } 14 $ Show that it is
↪  \mathrm{(E)}\ 14.-/
theorem amc12a_2003_p5 (A M C : ℕ) (h0 : A ≤ 9 ∧ M ≤ 9 ∧ C ≤ 9)
    (h1 : Nat.ofDigits 10 [0, 1, C, M, A] + Nat.ofDigits 10 [2, 1, C, M, A] = 123422) :
    A + M + C = 14 := by
      rcases h0 with ⟨hA, hM, hC⟩
      have hC1 : Nat.ofDigits 10 [0, 1, C, M, A] = 10 + 100 * C + 1000 * M + 10000 * A := by
        simp [Nat.ofDigits]
        ring
      have hC1' : Nat.ofDigits 10 [2, 1, C, M, A] = 12 + 100 * C + 1000 * M + 10000 * A := by
        simp [Nat.ofDigits]
        ring
      rw [hC1, hC1'] at h1
      have hC2 : C + 10 * M + 100 * A = 617 := by
        omega
      omega
```

Listing C: final local-reduced proof for `amc12a_2003_p5.lean`.

```
import Mathlib


open BigOperators Real Nat Topology Rat

/-- The sum of the two 5-digit numbers $AMC10$ and $AMC12$ is $123422$. What is $A+M+C$?

$ \mathrm{(A) \ } 10\qquad \mathrm{(B) \ } 11\qquad \mathrm{(C) \ } 12\qquad \mathrm{(D) \ } 13\qquad \mathrm{(E) \ } 14 $ Show that it is
↪  \mathrm{(E)}\ 14.-/
theorem amc12a_2003_p5 (A M C : ℕ) (h0 : A ≤ 9 ∧ M ≤ 9 ∧ C ≤ 9)
    (h1 : Nat.ofDigits 10 [0, 1, C, M, A] + Nat.ofDigits 10 [2, 1, C, M, A] = 123422) :
    A + M + C = 14 := by
      rcases h0 with ⟨hA, hM, hC⟩
      rw [show Nat.ofDigits 10 [0, 1, C, M, A] + Nat.ofDigits 10 [2, 1, C, M, A] = 22 + 200 * C + 2000 * M + 20000 * A from by
        simp [Nat.ofDigits]; ring] at h1
      omega
```

Listing D: original verified input for `imo2020_p2_algebra.lean` (HILBERT).

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat Polynomial
theorem hpos_d_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < d := by
  exact hd0

theorem hlt_b_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : b < 1 := by
  have h2 : b < 1 := by
    nlinarith [hba, hcb, hdc, hd0, h1]
  exact h2

theorem hpos_b_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < b := by
  have h2 : 0 < b := by
    -- Using the given conditions and linear arithmetic, we derive that b must be positive.
    nlinarith [hd0, hdc, hcb, hba, h1]
  exact h2

theorem hlt_d_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : d < 1 := by
  have h2 : d < 1 := by
    -- Using the given conditions and linear arithmetic, we can deduce that d < 1.
    nlinarith [hba, hcb, hdc, hd0, h1]
  exact h2

theorem hlt_c_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : c < 1 := by
  have : c < 1 := by
    -- Using the given conditions and the sum of the variables equals 1, we derive that c must be less than 1.
    nlinarith [hba, hcb, hdc, hd0, h1]
```

```
  exact this

theorem hL_gt_one_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 1 < a + 2 * b + 3 * c + 4 * d := by
  have h2 : a + 2 * b + 3 * c + 4 * d > 1 := by
    nlinarith [h1, hba, hcb, hdc, hd0]
  linarith

theorem hpos_a_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < a := by
  have h2 : a ≥ 0 := by
    nlinarith
  have h3 : b ≥ 0 := by
    nlinarith
  have h4 : c ≥ 0 := by
    nlinarith
  have h5 : d ≥ 0 := by
    nlinarith
  nlinarith [h2, h3, h4, h5, hd0, hdc, hcb, hba, h1]

theorem hpos_c_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < c := by
  have h2 : 0 < a + b + c := by
    nlinarith [hd0, hdc, hcb, hba, h1]
  have h3 : 0 < a + b := by
    nlinarith [hd0, hdc, hcb, hba, h1]
  have h4 : 0 < a := by
    nlinarith [hd0, hdc, hcb, hba, h1]
  nlinarith [hd0, hdc, hcb, hba, h1]

theorem hL_lt_four_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : a + 2 * b + 3 * c + 4 * d < 4 := by
  have h2 := mul_nonneg (sub_nonneg.mpr hdc) (sub_nonneg.mpr hcb)
  have h3 := mul_nonneg (sub_nonneg.mpr hcb) (sub_nonneg.mpr hba)
  have h4 := mul_nonneg (sub_nonneg.mpr hba) (sub_nonneg.mpr hdc)
  nlinarith [mul_self_nonneg (a - b), mul_self_nonneg (b - c), mul_self_nonneg (c - d),
    mul_self_nonneg (d - a)]

theorem hlt_a_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : a < 1 := by
  have h_sum_pos : 0 < b + c + d := by
    nlinarith [hd0, hdc, hcb, hba]

  have h_a : a = 1 - (b + c + d) := by
    nlinarith [h1]

  have h_a_lt_1 : a < 1 := by
    have h₂ : a = 1 - (b + c + d) := h_a
    have h₃ : 0 < b + c + d := h_sum_pos
    nlinarith

  exact h_a_lt_1

theorem h_prod_lt_one_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  have h2 : 0 < a := by linarith
  have h3 : 0 < b := by linarith
  have h4 : 0 < c := by linarith
  have h5 : 0 < d := by linarith
  have h6 : 0 < a * b := by positivity
  have h7 : 0 < a * b * c := by positivity
  have h8 : 0 < a * b * c * d := by positivity
  -- Use the fact that the product of numbers less than 1 is less than 1
  have h9 : a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
    have h10 : a ^ a < 1 := by
      apply Real.rpow_lt_one (by linarith) (by linarith)
      nlinarith
    have h11 : b ^ b < 1 := by
      apply Real.rpow_lt_one (by linarith) (by linarith)
      nlinarith
    have h12 : c ^ c < 1 := by
      apply Real.rpow_lt_one (by linarith) (by linarith)
      nlinarith
    have h13 : d ^ d < 1 := by
      apply Real.rpow_lt_one (by linarith) (by linarith)
      nlinarith
    calc
      a ^ a * b ^ b * c ^ c * d ^ d < 1 * 1 * 1 * 1 := by gcongr <;> assumption
      _ = 1 := by norm_num
  exact h9

theorem weighted_AM_GM_bound_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) :
    a ^ a * b ^ b * c ^ c * d ^ d ≤ ((a + 2*b + 3*c + 4*d) / (a + b + c + d)) ^ (a + b + c + d) := by
  have h_main : a ^ a * b ^ b * c ^ c * d ^ d ≤ 1 := by
    have h₂ : a ≥ 0 := by linarith
    have h₃ : b ≥ 0 := by linarith
    have h₄ : c ≥ 0 := by linarith
    have h₅ : d ≥ 0 := by linarith
    have h₆ : a ≤ 1 := by linarith
    have h₇ : b ≤ 1 := by linarith
    have h₈ : c ≤ 1 := by linarith
    have h₉ : d ≤ 1 := by linarith
    have h₁₀ : a ^ a ≤ 1 := by
      -- Use the fact that for x in [0,1], x^x ≤ 1
      have h₁₀₁ : a ≥ 0 := by linarith
      have h₁₀₂ : a ≤ 1 := by linarith
      exact Real.rpow_le_one (by linarith) (by linarith) (by linarith)
    have h₁₁ : b ^ b ≤ 1 := by
      -- Use the fact that for x in [0,1], x^x ≤ 1
      have h₁₁₁ : b ≥ 0 := by linarith
      have h₁₁₂ : b ≤ 1 := by linarith
      exact Real.rpow_le_one (by linarith) (by linarith) (by linarith)
    have h₁₂ : c ^ c ≤ 1 := by
      -- Use the fact that for x in [0,1], x^x ≤ 1
      have h₁₂₁ : c ≥ 0 := by linarith
      have h₁₂₂ : c ≤ 1 := by linarith
      exact Real.rpow_le_one (by linarith) (by linarith) (by linarith)
    have h₁₃ : d ^ d ≤ 1 := by
      -- Use the fact that for x in [0,1], x^x ≤ 1
      have h₁₃₁ : d ≥ 0 := by linarith
      have h₁₃₂ : d ≤ 1 := by linarith
      exact Real.rpow_le_one (by linarith) (by linarith) (by linarith)
    calc
      a ^ a * b ^ b * c ^ c * d ^ d ≤ 1 * 1 * 1 * 1 := by
        gcongr <;> assumption
      _ = 1 := by norm_num

  have h_rhs : ((a + 2*b + 3*c + 4*d) / (a + b + c + d)) ^ (a + b + c + d) = a + 2*b + 3*c + 4*d := by
    have h₂ : a + b + c + d = 1 := h1
    rw [h₂]
    <;> simp [h₂]
    <;> ring_nf
    <;> norm_num
    <;> nlinarith

  have h_final : a ^ a * b ^ b * c ^ c * d ^ d ≤ ((a + 2*b + 3*c + 4*d) / (a + b + c + d)) ^ (a + b + c + d) := by
    have h₃ : a ^ a * b ^ b * c ^ c * d ^ d ≤ 1 := h_main
    have h₄ : ((a + 2*b + 3*c + 4*d) / (a + b + c + d)) ^ (a + b + c + d) = a + 2*b + 3*c + 4*d := by
```

```
      rw [h_rhs]
    have h5 : a + 2*b + 3*c + 4*d > 0 := by
      nlinarith
    have h6 : a ^ a * b ^ b * c ^ c * d ^ d ≤ a + 2*b + 3*c + 4*d := by
      have h7 : a ^ a * b ^ b * c ^ c * d ^ d ≤ 1 := h_main
      have h8 : 1 ≤ a + 2*b + 3*c + 4*d := by
        nlinarith
      nlinarith
    linarith

  exact h_final

theorem ha_nonneg_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 ≤ a := by
  have h2 : 0 ≤ a := by
    nlinarith [hba, hcb, hdc, hd0, h1]
  exact h2

theorem hb_nonneg_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 ≤ b := by
  have h1 : b ≥ 0 := by
    nlinarith [hba, hcb, hdc, hd0, h1]
  nlinarith

theorem hc_lt_one_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : c < 1 := by
  have h2 : a + b + c + d = 1 := h1
  have h3 : a + b + c < 1 := by linarith
  have h4 : c < 1 := by linarith
  exact h4

theorem hd_lt_one_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : d < 1 := by
  have h2 : d < 1 := by
    -- Using the given conditions and the sum equation, we derive that d must be less than 1.
    nlinarith [hdc, hcb, hba, h1]
  exact h2

theorem hb_lt_one_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : b < 1 := by
  have h2 : b < 1 := by
    have h3 : a + b + c + d = 1 := h1
    have h4 : b < 1 := by
      nlinarith [hba, hcb, hdc, hd0, h3]
    exact h4
  exact h2

theorem hd_nonneg_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 ≤ d := by
  have h2 : 0 ≤ a := by
    nlinarith
  have h3 : 0 ≤ b := by
    nlinarith
  have h4 : 0 ≤ c := by
    nlinarith
  have h5 : 0 ≤ d := by
    nlinarith
  exact h5

theorem hb_pos_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < b := by
  by_contra! hb
  have h2 : a + b + c + d = 1 := h1
  have h3 : 0 < d := hd0
  have h4 : d ≤ c := hdc
  have h5 : c ≤ b := hcb
  have h6 : b ≤ a := hba
  have h7 : b ≤ 0 := by linarith
  have h8 : a + b + c + d = 1 := h1
  nlinarith [h3, h4, h5, h6, h7, h8]

theorem ha_pos_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < a := by
  have h_a_pos : 0 < a := by
    by_contra h
    have h2 : a ≤ 0 := by linarith
    have h3 : b ≤ 0 := by linarith
    have h4 : c ≤ 0 := by linarith
    have h5 : d ≤ 0 := by linarith
    have h6 : 0 < d := by linarith
    have h7 : d > 0 := by linarith
    linarith
  exact h_a_pos

theorem hc_pos_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 < c := by
  have h2 : 0 < c := by
    -- Use the given conditions to derive that c > 0
    linarith [hdc, hd0]
  exact h2

theorem ha_lt_one_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : a < 1 := by
  have h2 : a < 1 := by
    nlinarith [h1, hba, hcb, hdc, hd0]
  exact h2

theorem hc_nonneg_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) : 0 ≤ c := by
  have h2 : c ≥ 0 := by
    by_contra h
    have h3 : c < 0 := by linarith
    have h4 : d ≤ c := hdc
    have h5 : d > 0 := hd0
    have h6 : d ≤ c := hdc
    have h7 : c < 0 := h3
    have h8 : d < 0 := by linarith
    linarith
  linarith

theorem h_main_ineq_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (h_amgm : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) (h_TS : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1) : (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) < 1 := by
  have h_a : 0 ≤ a := by linarith
  have h_b : 0 ≤ b := by linarith
  have h_c : 0 ≤ c := by linarith
  have h_d : 0 ≤ d := by linarith
  have h_ab : 0 ≤ a * b := by positivity
  have h_ac : 0 ≤ a * c := by positivity
  have h_ad : 0 ≤ a * d := by positivity
  have h_bc : 0 ≤ b * c := by positivity
  have h_bd : 0 ≤ b * d := by positivity
  have h_cd : 0 ≤ c * d := by positivity
```

```
  have h_abcd : 0 ≤ a * b * c * d := by positivity
  have h_a_b_c_d : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by linarith
  have h_ineq_1 : (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) ≤ (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) := by
    gcongr
    <;>
    nlinarith [h_a, h_b, h_c, h_d, h_ab, h_ac, h_ad, h_bc, h_bd, h_cd, h_abcd, h_a_b_c_d]
  nlinarith [h_ineq_1, h_TS]

theorem h_final_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (h_main_ineq : (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) < 1) : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  have h_final : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
    have h2 : (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) < 1 := h_main_ineq
    have h3 : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d = (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) := by
      ring
    rw [h3] at *
    exact h2
  exact h_final

theorem h_amgm_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1)
    (ha_nonneg : 0 ≤ a) (hb_nonneg : 0 ≤ b) (hc_nonneg : 0 ≤ c) (hd_nonneg : 0 ≤ d) : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by
  -- Apply the weighted AM-GM inequality directly with a, b, c, d as both weights and values
  have hsum : a + b + c + d = 1 := h1
  have h_amgm := Real.geom_mean_le_arith_mean4_weighted ha_nonneg hb_nonneg hc_nonneg hd_nonneg
    ha_nonneg hb_nonneg hc_nonneg hd_nonneg hsum
  -- The theorem gives: a^a * b^b * c^c * d^d ≤ a * a + b * b + c * c + d * d
  -- But a * a = a^2, etc., so we can rewrite the right-hand side
  simpa [mul_comm, pow_two] using h_amgm

theorem hz_nonneg_h_TS_main_inequality_imo2020_p2_algebra (c d : ℝ) (hdc : d ≤ c) : 0 ≤ c - d := by
  have h : 0 ≤ c - d := by
    -- Use the given condition that d ≤ c to show that c - d ≥ 0
    linarith
  -- Use the established inequality to conclude the proof
  exact h

theorem hx_nonneg_h_TS_main_inequality_imo2020_p2_algebra (a b : ℝ) (hba : b ≤ a) : 0 ≤ a - b := by
  have h : 0 ≤ a - b := by
    -- Use the given condition that b ≤ a to show that a - b ≥ 0
    linarith
  -- Since we have shown that a - b ≥ 0, we can conclude that 0 ≤ a - b
  exact h

theorem hy_nonneg_h_TS_main_inequality_imo2020_p2_algebra (b c : ℝ) (hcb : c ≤ b) : 0 ≤ b - c := by
  have h : 0 ≤ b - c := by
    -- Use the given inequality c ≤ b to show that b - c is non-negative.
    linarith
  -- The result follows directly from the given inequality and the properties of real numbers.
  exact h

theorem hd_eq_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) : d = 1 - a - b - c := by
  have h2 : d = 1 - a - b - c := by
    -- We need to show that d = 1 - a - b - c.
    -- Start by isolating d on one side of the equation.
    linarith
  -- The above step uses linear arithmetic to solve for d, given the equation a + b + c + d = 1.
  exact h2

theorem hS2_eq_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 = a ^ 2 + b ^ 2 + c ^ 2 + (1 - a - b - c) ^ 2 := by
  have h2 : d = 1 - a - b - c := by linarith
  rw [h2]
  <;> ring_nf
  <;> linarith

theorem hT_eq_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) : a + 2 * b + 3 * c + 4 * d = 4 - 3 * a - 2 * b - c := by
  have h2 : a = 1 - b - c - d := by linarith
  rw [h2]
  ring_nf
  <;> nlinarith [sq_nonneg (b + c + d - 1), sq_nonneg (b + c + d - 2), sq_nonneg (b - c), sq_nonneg (c - d), sq_nonneg (d - b)]

theorem h_case_A_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (a = b ∧ b = c ∧ c = d) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
  rintro ⟨h₁, h₂, h₃⟩
  have h₄ : a = d := by linarith
  have h₅ : b = d := by linarith
  have h₆ : c = d := by linarith
  subst_vars
  nlinarith [mul_pos hd0 (sub_pos.mpr ha_lt_one), mul_pos hd0 (sub_pos.mpr hb_lt_one),
    mul_pos hd0 (sub_pos.mpr hc_lt_one), mul_pos hd0 (sub_pos.mpr hd_lt_one)]

theorem h_case_B_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (a = b ∧ b = c ∧ c > d) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
  rintro ⟨h1a, h1b, h1c⟩
  have h2 : a = b := by linarith
  have h3 : b = c := by linarith
  have h4 : c > d := by linarith
  rw [h2, h3] at h1 ⊢
  have h5 : d < 1 := by linarith
  have h6 : d > 0 := by linarith
  nlinarith [mul_pos h6 (by nlinarith : (0 : ℝ) < 1 - d)]

theorem hT_eq2_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) : a + 2 * b + 3 * c + 4 * d = 1 + 6 * d + (b - c) + 3 * (c - d) := by
  have h2 : a = 1 - b - c - d := by
    linarith [h1]
  simp_all only [h2, mul_sub, mul_one, mul_add, mul_comm]
  ring_nf
  <;> linarith

theorem hS2_lower_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (1/4 : ℝ) ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by
  have h₁ : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 ≥ 1 / 4 := by
    nlinarith [sq_nonneg (a - b), sq_nonneg (b - c), sq_nonneg (c - d), sq_nonneg (d - a),
      sq_nonneg (a + b - c - d), sq_nonneg (a + c - b - d), sq_nonneg (a + d - b - c),
      sq_nonneg (b + c - a - d), sq_nonneg (b + d - a - c), sq_nonneg (c + d - a - b)]
  linarith

theorem hx_eq_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) : a - b = 1 - 4 * d - 2 * (b - c) - 3 * (c - d) := by
  have h_main : a - b = 1 - 4 * d - 2 * (b - c) - 3 * (c - d) := by
    have h2 : a - b = 1 - 4 * d - 2 * (b - c) - 3 * (c - d) := by
      -- Simplify the right-hand side by expanding and rearranging terms
      simp [mul_comm, mul_assoc, mul_left_comm]
      -- Use linear arithmetic to verify the equality
      linarith
    -- The result follows directly from the simplified form
    linarith
  -- The final result is obtained from the main equality
  exact h_main
```

```
theorem h_case_C_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (b = c ∧ c = d ∧ a > b) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
  intro h
  have hb : b = c := h.1
  have hc : c = d := h.2.1
  have ha_gt_b : a > b := h.2.2
  have h_a : a = 1 - 3 * b := by linarith
  have hb_lt_one_fourth : b < 1 / 4 := by linarith
  have h_main : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    rw [h_a, hb, hc]
    norm_num
    nlinarith [sq_nonneg (b - 1 / 4), sq_nonneg (b + 1 / 4), sq_nonneg (b - 1 / 11), mul_pos hd0 (by linarith : (0 : ℝ) < 4)]
  exact h_main

theorem hS2_upper_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (h1 : a + b + c + d = 1) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 ≤ 1 := by
  have h_a_nonneg : 0 ≤ a := by
    nlinarith [hba, hcb, hdc, hd0]

  have h_b_nonneg : 0 ≤ b := by
    nlinarith [hba, hcb, hdc, hd0]

  have h_c_nonneg : 0 ≤ c := by
    nlinarith [hba, hcb, hdc, hd0]

  have h_d_nonneg : 0 ≤ d := by
    nlinarith [hba, hcb, hdc, hd0]

  have h_a_le_one : a ≤ 1 := by
    nlinarith [ha_lt_one, hb_lt_one, hc_lt_one, hd_lt_one, h1]

  have h_b_le_one : b ≤ 1 := by
    nlinarith [ha_lt_one, hb_lt_one, hc_lt_one, hd_lt_one, h1]

  have h_c_le_one : c ≤ 1 := by
    nlinarith [ha_lt_one, hb_lt_one, hc_lt_one, hd_lt_one, h1]

  have h_d_le_one : d ≤ 1 := by
    nlinarith [ha_lt_one, hb_lt_one, hc_lt_one, hd_lt_one, h1]

  have h_main : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 ≤ a + b + c + d := by
    have h2 : a ^ 2 ≤ a := by
      nlinarith [sq_nonneg (a - 1)]
    have h3 : b ^ 2 ≤ b := by
      nlinarith [sq_nonneg (b - 1)]
    have h4 : c ^ 2 ≤ c := by
      nlinarith [sq_nonneg (c - 1)]
    have h5 : d ^ 2 ≤ d := by
      nlinarith [sq_nonneg (d - 1)]
    nlinarith

  have h_final : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 ≤ 1 := by
    nlinarith

  exact h_final

theorem hd_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : 0 < d := by
  exact hd0

theorem ha_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) (hc_pos : 0 < c) (hb_pos : 0 < b) : 0 < a := by
  have h2 : 0 < a := by
    -- Use the given conditions and linear arithmetic to prove that a > 0
    nlinarith [hba, hcb, hdc, hd0, hb_pos, hc_pos, hd_pos]
  exact h2

theorem hb_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) (hc_pos : 0 < c) : 0 < b := by
  by_contra h
  have h2 : b ≤ 0 := by linarith
  have h3 : a + b + c + d = 1 := by linarith
  have h4 : a + c + d ≤ 1 := by linarith
  have h5 : a + b + c ≤ 1 := by linarith
  have h6 : a + b + c + d ≤ 1 := by linarith
  have h7 : a + b + c + d = 1 := by linarith
  have h8 : a + b + c + d ≤ 1 := by linarith
  have h9 : a + b + c + d = 1 := by linarith
  nlinarith [ha_lt_one, hb_lt_one, hc_lt_one, hd_lt_one, hd0, hc_pos, hdc, hcb, hba, h1, h2, h3, h4, h5, h6, h7, h8, h9]

theorem hS_bound_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) (hc_pos : 0 < c) (hb_pos : 0 < b) (ha_pos : 0 < a) (S : ℝ) (hS_def : S = a + 2 * b + 3 * c + 4 * d) (T : ℝ) (hT_def : T = a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) (hT_lt_one : T < 1) : S ≤ (5 : ℝ) / 2 := by
  have h₁ := mul_nonneg (sub_nonneg.mpr hba) (sub_nonneg.mpr hcb)
  have h₂ := mul_nonneg (sub_nonneg.mpr hcb) (sub_nonneg.mpr hdc)
  have h₃ := mul_nonneg (sub_nonneg.mpr hdc) (sub_nonneg.mpr hba)
  have h₄ := mul_nonneg (sub_nonneg.mpr hba) (sub_nonneg.mpr hdc)
  have h₅ := mul_nonneg (sub_nonneg.mpr hcb) (sub_nonneg.mpr hba)
  have h₆ := mul_nonneg (sub_nonneg.mpr hdc) (sub_nonneg.mpr hcb)
  nlinarith [sq_nonneg (a - 1 / 2), sq_nonneg (b - 1 / 2), sq_nonneg (c - 1 / 2), sq_nonneg (d - 1 / 2)]

theorem hc_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) : 0 < c := by
  have h_main : 0 < c := by
    by_contra h
    have h₂ : c ≤ 0 := by linarith
    have h₃ : d ≤ 0 := by
      nlinarith
    have h₄ : d > 0 := hd0
    nlinarith
  exact h_main

theorem h_main_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) (hc_pos : 0 < c) (hb_pos : 0 < b) (ha_pos : 0 < a) (S : ℝ) (hS_def : S = a + 2 * b + 3 * c + 4 * d) (T : ℝ) (hT_def : T = a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) (hT_lt_one : T < 1) (hS_bound : S ≤ (5 : ℝ) / 2) : S * T < 1 := by
  have h₁ : 0 < a * d := mul_pos ha_pos hd_pos
  have h₂ : 0 < b * c := mul_pos hb_pos hc_pos
  have h₃ : 0 < c * d := mul_pos hc_pos hd_pos
  have h₄ : 0 < a * b := mul_pos ha_pos hb_pos
  have h₅ : 0 < a * c := mul_pos ha_pos hc_pos
  have h₆ : 0 < b * d := mul_pos hb_pos hd_pos
  nlinarith [mul_nonneg (sub_nonneg.mpr hdc) (sub_nonneg.mpr hcb), mul_nonneg (sub_nonneg.mpr hcb) (sub_nonneg.mpr hba), mul_nonneg (sub_nonneg.mpr hba) (sub_nonneg.mpr hdc),
    mul_nonneg (sub_nonneg.mpr hdc) (sub_nonneg.mpr hba), mul_nonneg (sub_nonneg.mpr hcb) (sub_nonneg.mpr hdc), mul_nonneg (sub_nonneg.mpr hba) (sub_nonneg.mpr hcb)]

theorem hT_lt_one_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) (hd_pos : 0 < d) (hc_pos : 0 < c) (hb_pos : 0 < b) (ha_pos : 0 < a) (S : ℝ) (hS_def : S = a + 2 * b + 3 * c + 4 * d) (T : ℝ) (hT_def : T = a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) : T < 1 := by
  have h_a2 : a ^ 2 < a := by
```

```
    nlinarith [sq_pos_of_pos ha_pos, ha_lt_one]
  have h_b2 : b ^ 2 < b := by
    nlinarith [sq_pos_of_pos hb_pos, hb_lt_one]
  have h_c2 : c ^ 2 < c := by
    nlinarith [sq_pos_of_pos hc_pos, hc_lt_one]
  have h_d2 : d ^ 2 < d := by
    nlinarith [sq_pos_of_pos hd_pos, hd_lt_one]
  nlinarith [h1, h_a2, h_b2, h_c2, h_d2]

theorem h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b
↪  + c + d = 1) (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c
↪  ^ 2 + d ^ 2) < 1 := by
  have hd_pos : 0 < d := hd_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one
  ↪  hd_lt_one
  have hc_pos : 0 < c := hc_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one
  ↪  hd_lt_one hd_pos
  have hb_pos : 0 < b := hb_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one
  ↪  hd_lt_one hd_pos hc_pos
  have ha_pos : 0 < a := ha_pos_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one
  ↪  hd_lt_one hd_pos hc_pos hb_pos
  set S := a + 2 * b + 3 * c + 4 * d with hS_def
  set T := a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 with hT_def
  have hT_lt_one : T < 1 := hT_lt_one_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one
  ↪  hc_lt_one hd_lt_one hd_pos hc_pos hb_pos ha_pos S hS_def T hT_def
  have hS_bound : S ≤ (5 : ℝ) / 2 := hS_bound_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one
  ↪  hb_lt_one hc_lt_one hd_lt_one hd_pos hc_pos hb_pos ha_pos S hS_def T hT_def hT_lt_one
  have h_main : S * T < 1 := h_main_h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one
  ↪  hc_lt_one hd_lt_one hd_pos hc_pos hb_pos ha_pos S hS_def T hT_def hT_lt_one hS_bound
  exact h_main

theorem h_TS_main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1)
↪  (ha_lt_one : a < 1) (hb_lt_one : b < 1) (hc_lt_one : c < 1) (hd_lt_one : d < 1) : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2)
↪  < 1 := by
  -- Step 1: Express T = a + 2b + 3c + 4d as 4 - 3a - 2b - c
  have hT_eq : a + 2 * b + 3 * c + 4 * d = 4 - 3 * a - 2 * b - c := by
    exact hT_eq_h_TS_main_inequality_imo2020_p2_algebra a b c d h1
  -- Step 2: Express d in terms of a, b, c
  have hd_eq : d = 1 - a - b - c := by
    exact hd_eq_h_TS_main_inequality_imo2020_p2_algebra a b c d h1
  -- Step 3: Express S₂ = a² + b² + c² + d² in terms of a, b, c
  have hS2_eq : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 = a ^ 2 + b ^ 2 + c ^ 2 + (1 - a - b - c) ^ 2 := by
    exact hS2_eq_h_TS_main_inequality_imo2020_p2_algebra a b c d h1
  -- Step 4: Introduce new variables x = a - b, y = b - c, z = c - d
  have hx_nonneg : 0 ≤ a - b := by
    exact hx_nonneg_h_TS_main_inequality_imo2020_p2_algebra a b hba
  have hy_nonneg : 0 ≤ b - c := by
    exact hy_nonneg_h_TS_main_inequality_imo2020_p2_algebra b c hcb
  have hz_nonneg : 0 ≤ c - d := by
    exact hz_nonneg_h_TS_main_inequality_imo2020_p2_algebra c d hdc
  -- Step 5: Express x in terms of d, y, z
  have hx_eq : a - b = 1 - 4 * d - 2 * (b - c) - 3 * (c - d) := by
    exact hx_eq_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hdc hcb hba
  -- Step 6: Express T in terms of d, y, z
  have hT_eq2 : a + 2 * b + 3 * c + 4 * d = 1 + 6 * d + (b - c) + 3 * (c - d) := by
    exact hT_eq2_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hdc hcb hba
  -- Step 7: Bound S₂ between 1/4 and 1
  have hS2_lower : (1/4 : ℝ) ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by
    exact hS2_lower_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hd0 hdc hcb hba ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  have hS2_upper : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 ≤ 1 := by
    exact hS2_upper_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hd0 hdc hcb hba ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Step 8: Case analysis for when some variables are equal
  -- Case A: a = b = c = d
  have h_case_A : (a = b ∧ b = c ∧ c = d) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    exact h_case_A_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hd0 hdc hcb hba ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Case B: a = b = c > d
  have h_case_B : (a = b ∧ b = c ∧ c > d) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    exact h_case_B_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hd0 hdc hcb hba ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Case C: b = c = d < a
  have h_case_C : (b = c ∧ c = d ∧ a > b) → (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    exact h_case_C_h_TS_main_inequality_imo2020_p2_algebra a b c d h1 hd0 hdc hcb hba ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Step 9: Show that the maximum occurs at a boundary
  have h_boundary_max : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    exact h_boundary_max_h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Final conclusion
  exact h_boundary_max

theorem main_inequality_imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) :
    (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  -- Step 1: Show all variables are positive and less than 1
  have ha_pos : 0 < a := ha_pos_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hb_pos : 0 < b := hb_pos_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hc_pos : 0 < c := hc_pos_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have ha_lt_one : a < 1 := ha_lt_one_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hb_lt_one : b < 1 := hb_lt_one_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hc_lt_one : c < 1 := hc_lt_one_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hd_lt_one : d < 1 := hd_lt_one_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 2: Show exponents are nonnegative
  have ha_nonneg : 0 ≤ a := ha_nonneg_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hb_nonneg : 0 ≤ b := hb_nonneg_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hc_nonneg : 0 ≤ c := hc_nonneg_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hd_nonneg : 0 ≤ d := hd_nonneg_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 3: Apply weighted AM-GM inequality (non-strict version)
  have h_amgm : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 :=
    h_amgm_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_nonneg hb_nonneg hc_nonneg hd_nonneg
  -- Step 4: Define T and S
  let T := a + 2 * b + 3 * c + 4 * d
  let S := a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2
  -- Step 5: Show T * S < 1
  have h_TS : T * S < 1 := h_TS_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 ha_lt_one hb_lt_one hc_lt_one hd_lt_one
  -- Step 6: Combine the inequalities
  have h_main_ineq : T * (a ^ a * b ^ b * c ^ c * d ^ d) < 1 :=
    h_main_ineq_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 h_amgm h_TS
  -- Step 7: Rearrange to match the goal
  have h_final : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 :=
    h_final_main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1 h_main_ineq
  exact h_final

theorem imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a)
    (h1 : a + b + c + d = 1) : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  -- Step 1: Show all variables are positive
  have hpos_d : 0 < d := hpos_d_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hpos_c : 0 < c := hpos_c_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hpos_b : 0 < b := hpos_b_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hpos_a : 0 < a := hpos_a_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 2: Show all variables are less than 1
  have hlt_d : d < 1 := hlt_d_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hlt_c : c < 1 := hlt_c_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hlt_b : b < 1 := hlt_b_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  have hlt_a : a < 1 := hlt_a_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 3: Define L = a + 2b + 3c + 4d
  set L := a + 2 * b + 3 * c + 4 * d with hL_def
  -- Step 4: Show L > 1
  have hL_gt_one : 1 < L := hL_gt_one_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 5: Show a^a * b^b * c^c * d^d < 1
```

```
  have h_prod_lt_one : a ^ a * b ^ b * c ^ c * d ^ d < 1 := h_prod_lt_one_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 6: Show L < 4
  have hL_lt_four : L < 4 := hL_lt_four_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 7: Use weighted AM-GM bound
  have h_bound := weighted_AM_GM_bound_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  -- Step 8: Simplify using h1: a + b + c + d = 1
  rw [h1, div_one] at h_bound
  -- Step 9: Combine with L > 1 and product < 1
  have h_main : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 :=
    main_inequality_imo2020_p2_algebra a b c d hd0 hdc hcb hba h1
  exact h_main
```

Listing E: global restructuring output for `imo2020_p2_algebra.lean`.

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat Polynomial
theorem imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a)
    (h1 : a + b + c + d = 1) : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  have h2 : a > 0 := by linarith
  have h3 : b > 0 := by linarith
  have h4 : c > 0 := by linarith
  have h5 : d > 0 := hd0
  have hT_nonneg : 0 ≤ a + 2 * b + 3 * c + 4 * d := by positivity


  have hTS : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    nlinarith [sq_nonneg (a - b), sq_nonneg (b - c), sq_nonneg (c - d),
      mul_pos h2 h3, mul_pos h3 h4, mul_pos h4 h5, mul_pos h2 h4, mul_pos h2 h5, mul_pos h3 h5]

  have hAMGM : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by
    have ha_pos : 0 ≤ a := by linarith
    have hb_pos : 0 ≤ b := by linarith
    have hc_pos : 0 ≤ c := by linarith
    have hd_pos : 0 ≤ d := by linarith
    have := Real.geom_mean_le_arith_mean4_weighted ha_pos hb_pos hc_pos hd_pos ha_pos hb_pos hc_pos hd_pos h1
    simpa [pow_two] using this

  calc
    (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d
        = (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) := by ring
    _ ≤ (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) := by
          exact mul_le_mul_of_nonneg_left hAMGM hT_nonneg
    _ < 1 := hTS
```

Listing F: final verifier-selected proof for `imo2020_p2_algebra.lean`.

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat Polynomial
theorem imo2020_p2_algebra (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a)
    (h1 : a + b + c + d = 1) : (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d < 1 := by
  have h2 : a > 0 := by linarith
  have h3 : b > 0 := by linarith
  have h4 : c > 0 := by linarith
  have h5 : d > 0 := hd0
  have hT_nonneg : 0 ≤ a + 2 * b + 3 * c + 4 * d := by positivity


  have hTS : (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) < 1 := by
    nlinarith [sq_nonneg (a - b), sq_nonneg (b - c), sq_nonneg (c - d),
      mul_pos h2 h3, mul_pos h3 h4, mul_pos h4 h5, mul_pos h2 h4, mul_pos h2 h5, mul_pos h3 h5]

  have hAMGM : a ^ a * b ^ b * c ^ c * d ^ d ≤ a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 := by
    have ha_pos : 0 ≤ a := by linarith
    have hb_pos : 0 ≤ b := by linarith
    have hc_pos : 0 ≤ c := by linarith
    have hd_pos : 0 ≤ d := by linarith
    have := Real.geom_mean_le_arith_mean4_weighted ha_pos hb_pos hc_pos hd_pos ha_pos hb_pos hc_pos hd_pos h1
    simpa [pow_two] using this

  calc
    (a + 2 * b + 3 * c + 4 * d) * a ^ a * b ^ b * c ^ c * d ^ d
        = (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) := by ring
    _ ≤ (a + 2 * b + 3 * c + 4 * d) * (a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2) := by
          exact mul_le_mul_of_nonneg_left hAMGM hT_nonneg
    _ < 1 := hTS
```