MAREASON: ADAPTIVE RETRIEVAL AND DEEP PROOF PLANNING FOR MATHEMATICAL ANALYSIS IN LEAN 4

Anonymous authors
Paper under double-blind review

ABSTRACT

Formal theorem proving in mathematical analysis requires finding relevant results in Mathlib, Lean's mathematical library, and connecting them into a complete argument. Language-model provers can invoke unavailable library facts or fail to turn relevant premises into a complete proof. We present MAReason, which uses Lean verification to trigger escalation around a fixed OProver-8B. It first strengthens the premise context for whole-proof generation. Persistent verification failure triggers a change in proof granularity to independently grounded and verified obligations, followed by final composition and Lean verification. MAReason-Corpus provides domain-specific, Lean-verified supervision with benchmark-overlap removal. We evaluate on MA-ProofBench's 200 problems, comprising undergraduate analysis (Level I) and Ph.D.-qualifying analysis (Level II). With eight candidates per retrieval stage and eight Deep search trajectories, MAReason raises Pass@8 from 4% to 19% on Level I and from 1% to 4% on Level II relative to direct OProver-8B, using additional staged search. Mathlib hallucination fractions among failed candidates fall from 54.6% to 49.9% on Level I and from 39.0% to 35.9% on Level II. In a separate study, domain adaptation with MAReason-Corpus improves Level I performance on the base model but solves no Level II problems. These results support domain adaptation for easier analysis proofs and verified proof planning for harder obligations.

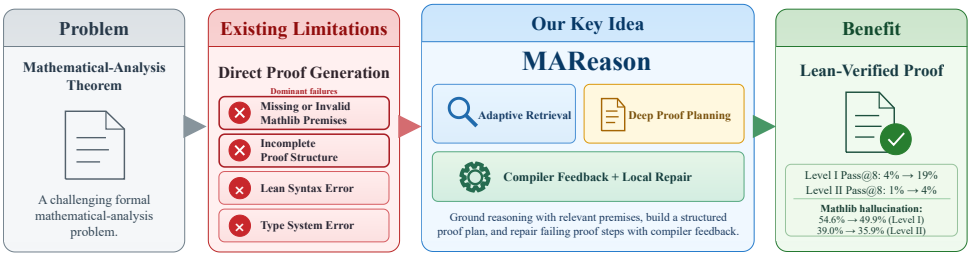


Figure 1: Key idea of MAReason. MAReason addresses the dominant failure modes in mathematical-analysis theorem proving through adaptive retrieval, deep proof planning, and compiler-guided local repair, producing Lean-verified proofs while improving proof success rates and reducing Mathlib hallucination.

1 INTRODUCTION

Formal theorem proving requires a proof assistant to check every step of a mathematical argument. In analysis, this means connecting specialized results in Mathlib, Lean's mathematical library, through long chains of intermediate claims. Language-model provers can fail at either task: they may invoke unavailable library facts or leave the argument unfinished. MA-ProofBench makes these difficulties concrete. Its 200 Lean 4 problems range from undergraduate exercises to Ph.D.-qualifying analysis, with Mathlib hallucination and incomplete proofs among the recurring failure modes (Pu et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Math-focused pretraining and reinforcement learning have improved informal reasoning (Shao et al., 2024; Yang et al., 2024; DeepSeek-AI, 2025; He et al., 2025). Formal provers additionally use retrieval, decomposition, and verifier feedback to find and repair proofs (Yang et al., 2023; Ren et al., 2025; Ma et al., 2026). These mechanisms address distinct needs: retrieval supplies library facts, while decomposition exposes intermediate claims. Both must ultimately yield a verified proof. Our central principle is verification-triggered escalation: first strengthen the premise context for whole-proof generation, then change the unit of proof search when that intervention remains unsuccessful (Figure 1).

MAREASON applies this principle with a fixed proof model. Standard Retrieval supplies a compact premise set; verification failure triggers sketch-guided Reasoning Retrieval. Remaining failures enter Deep Mode, which proposes two to four dependent auxiliary lemmas. Each lemma receives its own retrieval context and is proved or repaired using compiler feedback before final composition. Lean checks the proposed statements, local proofs, and assembled theorem at separate stages, producing independently grounded and verified obligations while preserving the target theorem and prover weights.

Domain-specific supervision offers a complementary way to address analysis proving. Formal evaluation now spans competition mathematics in miniF2F and PutnamBench (Zheng et al., 2021; Tsoukalas et al., 2024), undergraduate formalization in ProofNet, and broad Lean 4 coverage in FormalMATH (Azerbayev et al., 2023; Yu et al., 2025). TaoAnalysisBench and MA-ProofBench focus this evaluation on textbook and advanced analysis (TaoAnalysisBench; Pu et al., 2026). Training resources remain broad and heterogeneous. We therefore curate MAReason-Corpus around analysis subdomains, with Lean verification and normalized benchmark-overlap removal. A separate adaptation study tests whether this supervision addresses the same difficulties as inference-time retrieval and planning.

Our contributions are:

- **Verification-triggered escalation.** Lean verification governs a progression from premise-augmented whole-proof generation to finer-grained obligations. Each failed stage triggers the next intervention with a fixed target theorem and prover.
- **Change in proof granularity.** Deep Mode changes the unit of proof search through a Lean-checked blueprint of two to four dependent auxiliary lemmas. Each receives its own premises, proof search, and repair. Independently grounded and verified obligations are composed and checked in the original theorem context.
- **MAReason-Corpus.** We construct 309,953 Lean-verified theorem–proof pairs across 12 mathematical-analysis subdomains through domain filtering, deduplication, compilation, and benchmark decontamination. Controlled ablations show that domain-aligned data can be more effective than a substantially larger general Lean proof corpus.

2 RELATED WORK

2.1 FORMAL THEOREM PROVING WITH LANGUAGE MODELS

Math-focused pretraining and reinforcement learning with verifiable rewards advance mathematical reasoning (Azerbayev et al., 2024; Hu et al., 2025; Li et al., 2025). Lean provers additionally learn from proof-assistant execution. DeepSeek-Prover uses verifier feedback and reinforcement learning to improve formal proof generation (Xin et al., 2024; Ren et al., 2025). Goedel-Prover combines scaffolded proof synthesis with self-correction (Lin et al., 2025). Other systems explore different units of proof search: BFS-Prover searches over proof steps (Xin et al., 2025), Pythagoras-Prover focuses on efficient whole-proof generation (Leang et al., 2026), and OProver incorporates retrieval and compiler-guided repair into an agentic proving process (Ma et al., 2026). Verifier feedback identifies errors, while selecting useful premises and constructing the intermediate argument remain central tasks for the prover.

Training resources provide complementary forms of supervision. LeanWorkbook supplies autoformalized problems (Ying et al., 2024), while Lean-GitHub mines existing Lean repositories (Wu et al., 2024). Natural-language annotations in Herald and bootstrapped proofs in TheoremLlama further connect informal descriptions with formal reasoning (Gao et al., 2025; Wang et al., 2024b).

OProofs retains the proving process as well as successful proofs, including retrieval context, failed attempts, and compiler-guided repairs (Ma et al., 2026). These broad resources already contain analysis material. MAReason-Corpus curates domain-aligned supervision through filtering, compilation, and benchmark-overlap removal. A separate LoRA study (Hu et al., 2022) tests this supervision on MA-ProofBench's undergraduate and Ph.D.-qualifying analysis problems, distinguishing domain adaptation from the fixed-prover inference system.

2.2 Premise Selection and Retrieval-Augmented Proving

Premise selection connects a proof goal to useful results in a formal library. Lightweight learned suggestions support this task in Lean (Piotrowski et al., 2023), while LeanDojo's ReProver learns retrieval from dependency annotations and couples retrieved premises with tactic generation (Yang et al., 2023). Lean Copilot provides interactive premise and proof-step suggestions (Song et al., 2024). Other approaches improve the retrieval representation: Magnushammer uses contrastive transformer retrieval in Isabelle (Mikuła et al., 2024), and structural retrieval combines textual embeddings with dependency graphs (Petrovčič et al., 2025). Lean Finder aligns queries with formal statements using intent-oriented synthetic queries and preference feedback (Lu et al., 2025).

LeanSearch v2 is the closest retrieval foundation for our system (Gao et al., 2026). Its Standard mode combines embeddings and reranking over informalized Mathlib declarations; its Reasoning mode uses proof sketches, multiple queries, and iterative filtering and reflection to retrieve premises for an entire theorem. The sketch-driven mode can express intermediate mathematical needs that are absent from a direct statement query. MAReason uses verification-triggered escalation to change the unit of proof search from the whole theorem to obligations with their own premise contexts. Accessibility and relevance do not establish applicability under the target's hypotheses: retrieval constructs context, while Lean proof verification determines success and escalation.

2.3 Proof Planning and Verification-Guided Reasoning

Proof planning introduces intermediate structure between a theorem and its complete formal proof. Draft, Sketch, and Prove turns informal arguments into formal sketches whose gaps an automated prover fills (Jiang et al., 2023). LEGO-Prover organizes proving around reusable verified lemmas and grows a library that supports subsequent problems (Wang et al., 2024a). DeepSeek-Prover-V2 uses recursive decomposition for data synthesis and reinforcement learning (Ren et al., 2025). LeanSTaR interleaves informal thoughts with formal tactics and learns from verified sampled proofs (Lin et al., 2024). Intermediate reasoning thus supports sketch completion, library construction, and prover training.

Verifier-guided search and repair use execution feedback to revise the next proof attempt. HyperTree Proof Search couples search with online learning (Lample et al., 2022), and DeepSeek-Prover-V1.5 incorporates proof-assistant feedback into Monte-Carlo tree search (Xin et al., 2024). Goedel-Prover-V2 uses compiler-based correction (Lin et al., 2025), while OProver combines retrieval with compiler-guided repair (Ma et al., 2026). MAReason applies repair locally to auxiliary obligations and separately checks auxiliary-statement legality, local proofs, and final composition: a well-formed statement need not be provable, and a verified lemma need not close the target.

MAReason emphasizes selective escalation with a fixed prover: verified decomposition is reserved for goals unresolved by retrieval-guided generation. Unlike reusable library growth (Wang et al., 2024a) or decomposition for prover training (Ren et al., 2025), MAReason uses verified decomposition selectively at inference time.

3 Method

3.1 MAReason-Corpus

MAReason-Corpus supports a controlled study of domain adaptation: it selects verified analysis proofs from heterogeneous Lean resources. We preserve each example's source, Lean statement, complete proof, and natural-language statement when available. Records with missing statements, missing proofs, or proof holes such as `sorry` and `admit` are discarded before domain filtering.

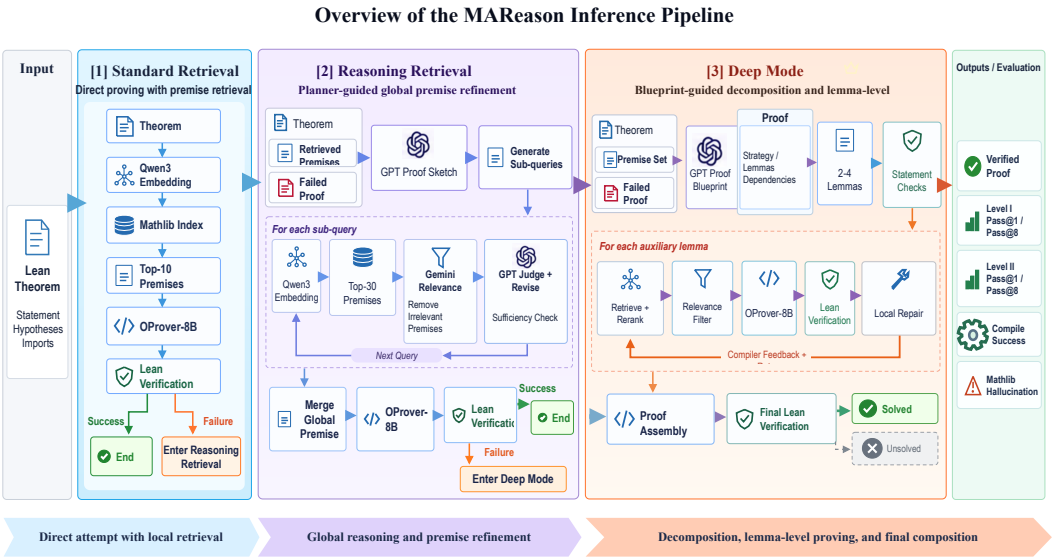


Figure 2: Overview of the MAReason framework. Standard Retrieval runs first; unresolved targets escalate to Reasoning Retrieval and then Deep Mode. A verified candidate prevents escalation to a later stage. In Deep Mode, the blueprint is checked before proof generation, each auxiliary lemma receives its own retrieval context, compiler errors drive local repair, and the assembled theorem undergoes final Lean verification.

Guided by the MSC-based taxonomy used by MA-ProofBench, which organizes analysis into six core categories and 27 subcategories (Pu et al., 2026), we adapt this structure into 12 training-oriented subdomains, including real analysis, limits and continuity, measure theory, integration, complex analysis, and functional analysis. Exact proof duplicates and repeated theorem statements are removed across sources before compilation.

Compilation in Lean 4.28.0 with the matched Mathlib environment checks that the retained proofs are executable. We then remove normalized exact-statement overlaps with all 200 MA-ProofBench targets. This procedure yields 309,953 unique, Lean-verified theorem–proof pairs. Natural-language statements are retained when supplied by the source but are not required; 9,390 examples contain one. OProofs contributes 87.6% of the final corpus, which motivates the matched-size and full-scale OProofs controls in Section 4.4. Full source counts, filtering statistics, and token-length coverage are reported in Appendix B.

The corpus supplies statement-to-proof supervision for the adaptation study. The inference system below uses unchanged OProver weights, allowing us to study retrieval and planning separately from parameter updates.

3.2 Problem Formulation and Overview

MAReason seeks a complete proof program p for a Lean statement x, preserving its imports and local assumptions. Let $V(p, x) \in \{0, 1\}$ denote Lean verification: $V(p, x) = 1$ when p proves the unchanged statement, contains no proof hole, and compiles in the target Lean and Mathlib environment. The proof model remains fixed. Lean verification supplies the routing signal: escalation first enriches the whole-proof premise context, then changes proof granularity.

Let $C_s(x) = (p_{s,1}, \ldots, p_{s,n})$ denote the candidate batch at stage $s \in \{1, 2, 3\}$, corresponding to Standard, Reasoning, and Deep. MAReason selects the first successful stage:

$$s^*(x) = \min\Big\{s : \max_{1 \le j \le n} V(p_{s,j}, x) = 1\Big\}. \tag{1}$$

Thus a verified proof ends escalation, while an all-failed batch triggers the next stage (Figure 2). If the set in Equation 1 is empty, the problem remains unsolved. Routing depends on verification outcomes, not inferred failure causes. For evaluation, we retain the first successful stage's full candidate batch.

3.3 HIERARCHICAL PREMISE RETRIEVAL

Standard Retrieval. The first stage tests whether a compact set of library facts is sufficient. A query combines the Lean statement with its informal description when available. Following LeanSearch v2 (Gao et al., 2026), an embedding model retrieves 50 Mathlib declarations and a reranker selects Top-10. Writing R_K for retrieval and reranking and A_x for accessibility filtering, the prover context is $P_{\mathrm{S}}(x) = A_x(R_{10}(x))$. The filter removes the target theorem, exact statement duplicates, duplicate declarations, and unavailable names. OProver receives the resulting premises and original theorem in the LeanSearch prover prompt; its candidates are normalized and checked by Lean.

Reasoning Retrieval. An unsuccessful first stage motivates a search for the ingredients of the whole argument. A reasoning model sketches a proof, derives queries for its steps, and revises the sketch when retrieved facts leave gaps. Each query retrieves and reranks 30 candidates before a separate model filters them for relevance. Cross-query rank aggregation, deduplication, and accessibility checks produce up to 30 global premises, with at most three revisions of the initial sketch. OProver then attempts the original theorem with this expanded context. The sketch guides premise search; OProver generates the Lean proof and Lean determines whether it succeeds.

Relationship to LeanSearch v2. MAReason's verification-triggered escalation moves retrieval from the whole theorem to individual obligations after whole-proof generation fails. Premise search uses LeanSearch v2's released embedding–reranking backend and decompose–retrieve–filter–judge loop (Gao et al., 2026), with analysis prompts, separate reasoning and filtering roles, and accessibility and target-leakage checks. Obligations receive separate retrieval contexts and verification before composition.

3.4 LEAN-CHECKED PROOF BLUEPRINTS

When whole-proof generation remains unsuccessful, Deep Mode changes the unit of proof search to auxiliary obligations. The reasoning model receives the unchanged theorem, accessible target-level premises, and Reasoning Retrieval trace. It proposes a *Proof Blueprint*: auxiliary claims, their dependencies, and a final step connecting them to the target. Formally,

$$B = \big(\sigma, \{(\ell_i, D_i, H_i)\}_{i=1}^{m}, b\big), \qquad 2 \le m \le 4, \quad D_i \subseteq \{1, \dots, i-1\}, \quad H_i \subseteq H_x. \tag{2}$$

Here σ is the strategy, ℓ_i an auxiliary proposition, D_i its dependencies, H_i the original hypotheses it uses from H_x, and b the final composition step. Each claim can depend only on earlier claims. This defines a directed acyclic graph and an order for proving the obligations, while restricting assumptions to the original theorem context.

The blueprint must pass structural and Lean checks before proof search begins. Structural checks reject malformed plans, invalid dependencies, new assumptions, and uses or copies of the target theorem. Deterministic code constructs Lean declarations from the proposed propositions and original context. Lean then checks each auxiliary statement, the final bridge statement, and the complete dependency graph in the target context. This establishes that the obligations are well formed; the next stage must still prove them. The precise schema and checks appear in Appendix A.

3.5 VERIFIED DECOMPOSITION, REPAIR, AND COMPOSITION

Deep Mode turns the blueprint into a sequence of smaller proving tasks. For each well-formed obligation, Reasoning Retrieval uses its statement and the overall strategy to find premises tailored to that step. OProver proves the obligations in dependency order, making verified earlier lemmas available to later ones. Each proof must compile both independently and inside the original target context with its verified dependencies. The second check ensures that a locally valid proof can actually be used in the intended composition.

A failed local proof is revised using its compiler diagnostic while keeping the retrieved premises fixed. For candidate $p_i^{(t)}$, diagnostic $e_i^{(t)}$, and premise set P_i, the repair step is

$$p_i^{(t+1)} \sim P_\theta\big(\cdot \mid \Gamma_i, \ell_i, P_i, p_i^{(t)}, e_i^{(t)}\big), \qquad t < 2, \tag{3}$$

where Γ_i contains the original context and verified dependencies, and θ remains fixed. Each node has at most four candidates, each with two repair rounds; both verification checks are repeated after repair. Once all obligations are proved, OProver generates a bridge proof connecting them to the original goal. Deterministic code assembles the verified auxiliary proofs as local `have` declarations followed by this bridge. The complete theorem must pass final Lean verification to count as a Deep Mode solution. Appendix A specifies caching, candidate retention, and verification details. Appendix D follows a concrete theorem through retrieval failures and successful decomposition.

4 EXPERIMENTS

We organize the evaluation around three questions.
1. Overall effectiveness. Does MAReason improve formal analysis proving over direct generation and retrieval-only systems? We compare proof success and Mathlib hallucination across the two benchmark levels.
2. Component contributions. How do retrieval, planning, and filtering affect proof completion? We examine stage-wise escalation and the separate planner and relevance-filter roles.
3. Domain-specific supervision. Is domain-aligned supervision more useful than a larger general proof corpus? We compare training corpora while holding the prover, adaptation procedure, and direct-evaluation protocol fixed.

4.1 EXPERIMENTAL SETUP

Benchmark and verification. MA-ProofBench (Pu et al., 2026) contains 100 undergraduate problems (Level I) and 100 Ph.D.-qualifying problems (Level II), allowing us to compare the same system across two levels of mathematical difficulty. A candidate succeeds only when the benchmark's Kimina reconstruction pipeline verifies a complete proof of the unchanged statement in Lean 4.28.0 with the matched Mathlib environment.

Pass@k and candidate accounting. Pass@k estimates the chance of finding a verified proof among k candidates. Following the benchmark evaluation, for $N = 100$ problems with n_i evaluated candidates and c_i verified successes per problem, we report

$$\widehat{\text{Pass@}k} = \frac{100}{N} \sum_{i=1}^{N} \left[1 - \frac{\binom{n_i - c_i}{k}}{\binom{n_i}{k}} \right], \qquad 1 \le k \le n_i, \tag{4}$$

where $\binom{a}{k} = 0$ for $a < k$. With $n_i = 8$, Pass@1 averages $c_i/8$, whereas Pass@8 is the percentage of problems with at least one verified candidate. Direct $n_i = 32$ runs use the same formula for $k \in \{1, 8, 32\}$. For MAReason, Equation 1 selects the first successful stage's full eight-candidate batch; if every stage fails, $c_i = 0$. Earlier failed batches and intermediate lemma attempts are excluded from n_i. Consequently, MAReason's Pass@1 measures success within the batch retained after routing, rather than a single end-to-end attempt. Its Pass@8 is the solve rate of the complete staged search.

Generation protocol. Proof generation uses unchanged OProver-8B weights, temperature 1.0, top-p 0.95, global seed 20260812, and at most 32,768 new tokens unless stated otherwise. GPT-5.6-sol (xhigh) supplies sketches, judgments, and blueprints; Gemini-3.7-Flash filters premise relevance. Each retrieval stage has eight candidates, and Deep Mode has eight trajectories with additional auxiliary attempts and repairs. This is a larger search budget than eight direct generations. Samples share cached retrieval and planning but use distinct deterministic seeds. Appendix A gives budgets, prompts, and API provenance.

Mathlib hallucination. Following Pu et al. (2026), we measure library-reference failures using H/F. Here H counts failed candidates whose compiler diagnostics identify absent, inaccessible, malformed, or incorrectly namespaced Lean/Mathlib declarations, fields, namespaces, or tactics; F counts all failed candidates. Attribution covers every failure. Resource limits and other errors without such diagnostics do not establish a hallucination, so the measure captures compiler-evidenced failures rather than every possible semantic error.

Table 1: MA-ProofBench results with $n = 8$. Pass@k is in percent; Solved counts verified problems. * marks benchmark-reported results (Pu et al., 2026); other rows use Kimina and Lean 4.28.0. MAReason's Pass@1 uses the retained stage's batch and Pass@8 measures staged solve rate, with more search than eight direct attempts. Higher is better. Across model/system rows, bold and underline mark the best and second-best distinct values; ties share markings. The final row gives gains over OProver-8B in percentage points and solved problems and is excluded from ranking.

	Level I			Level II		
Model / system	Pass@1	Pass@8	Solved	Pass@1	Pass@8	Solved
GPT-5.5 (xhigh)*	<u>6.500</u>	<u>16.000</u>	<u>16</u>	**1.750**	**5.000**	**5**
Gemini 3.1 Pro (High)*	4.380	13.000	13	<u>1.630</u>	**5.000**	**5**
Claude Sonnet 4.6 (High)*	3.500	6.000	6	0.500	3.000	3
DeepSeek-Prover-V2-7B	0.375	2.000	2	0.000	0.000	0
Goedel-Prover-V2-8B	1.750	3.000	3	0.000	0.000	0
OProver-8B-Base	2.125	5.000	5	0.000	0.000	0
OProver-8B	2.250	4.000	4	0.125	1.000	1
OProver-32B	2.750	5.000	5	0.375	1.000	1
MAReason-8B	**7.875**	**19.000**	**19**	1.250	<u>4.000</u>	<u>4</u>
Δ vs. OProver-8B	+5.625	+15.000	+15	+1.125	+3.000	+3

4.2 Main Results

Retrieval and verified planning extend the range of analysis problems solved by OProver-8B (Ma et al., 2026). On Level I, MAReason-8B achieves the highest solve rate in Table 1, exceeding the three benchmark-reported proprietary direct baselines, including GPT-5.5 (xhigh). This is a system-level gain with additional staged search. On Level II, MAReason-8B solves more problems than the locally evaluated direct provers, while GPT and Gemini retain higher scores. The harder level therefore remains a challenge even with verified decomposition.

Sampling budget also changes the direct-prover ranking (Figure 3(a)). OProver-32B leads at Pass@1 and Pass@8, whereas OProver-8B-Base covers more problems at Pass@32. More reliable individual attempts and broader coverage are therefore different strengths. On Level II, only OProver-8B and OProver-32B solve any problems in these direct runs. OProver-8B provides a smaller backbone with nonzero advanced-analysis performance, allowing us to study inference structure while fixing prover size.

4.3 Ablation Studies

Premise budget. Increasing the premise budget tests whether missing library facts alone explain the remaining failures. In Figure 3(b–c), Top-10 provides the best observed coverage and lowest hallucination fraction; larger sets do not improve solving. This motivates refining queries and proof structure. Top-100 also changes prompt rendering, so its effect cannot be attributed solely to premise count. Top-10 was selected on this benchmark, rather than independently tuned on held-out data.

System stages. The stage ablation separates premise grounding from proof construction (Table 2). Standard Retrieval improves both solving and library-reference accuracy. Reasoning Retrieval adds no solved problems and raises the Level I hallucination fraction: a broader, sketch-guided premise set alone does not resolve the remaining failures. Deep Planning and Repair then achieve the highest solve rate on both levels. Standard Retrieval still has the lowest Level I hallucination fraction, showing that successful proof construction and library-reference accuracy are distinct outcomes. The Deep gain reflects decomposition, repair, and their additional search budget jointly.

Reasoning/planning and filtering. We examine these roles on Hard-Dev, a separate set of 40 long theorem–proof examples from the fixed 2% MAReason-Corpus validation split. It covers all 12 domains and four sources, with no normalized statement overlap with MA-ProofBench. Statements

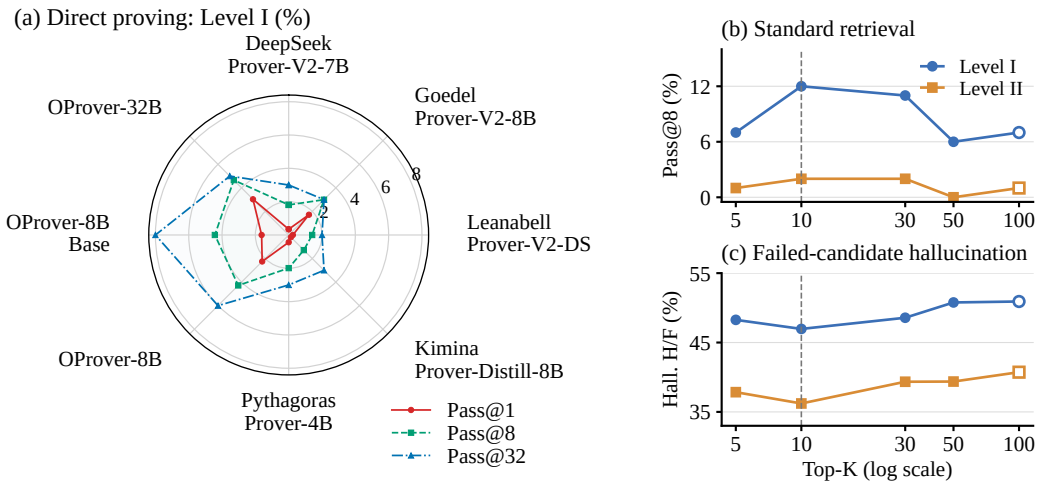


Figure 3: Direct proving and premise-budget ablation. (a) Level I direct $n = 32$ results (%): Pass@1/8/32 in red/green/blue. (b–c) OProver-8B Standard Retrieval, $n = 8$: success and full-audit hallucination $100H/F$ among failed candidates; dashed lines mark Top-10. Hollow Top-100 points use names-only prompts (Appendix C). Ten all-zero models are listed in Appendix C. Level II Pass@1/Pass@8/Pass@32 is nonzero only for OProver-8B (0.125/0.705/1.000) and OProver-32B (0.500/0.999/1.000).

Table 2: Controlled MAReason stage ablation with OProver-8B and $n = 8$. All metrics are percentages. Hall. is the Mathlib hallucination fraction H/F over all failed candidates. Column-best values are bold (including ties); higher is better for Pass@k and lower is better for Hall. Direct is a separate frozen run from the baseline in Table 1.

	Level I			Level II		
Method	Pass@1	Pass@8	Hall.	Pass@1	Pass@8	Hall.
Direct OProver-8B	2.500	4.000	54.615	0.125	1.000	39.049
+ Standard Top-10	3.375	12.000	**46.960**	0.250	2.000	36.216
+ Reasoning Retrieval	3.375	12.000	50.841	0.250	2.000	36.216
+ Deep Planning and Repair	**7.875**	**19.000**	49.932	**1.250**	**4.000**	**35.949**

and reference proofs compile; generation sees only the statements. Selection details appear in Appendix C. The GPT configurations replace both retrieval reasoning and blueprint generation, while the Gemini configurations replace premise filtering.

Table 3 shows higher auxiliary-proof and final success when GPT replaces the local reasoning/planning configuration (A→B). This comparison changes sketching, judging, blueprint generation, and their budgets together. Gemini filtering alone (A→C) gives the lowest hallucination fraction but solves fewer problems. Combining both changes yields the highest final solve rate, with a modest gain over GPT alone. Cleaner library references therefore do not by themselves ensure proof completion. Adding GPT improves both metrics from A→B, but increases solving and hallucination from C→D. The trade-off depends on the configuration and on which candidates remain unsuccessful.

4.4 Domain-Specific Data Ablation

The data ablation asks whether analysis-focused supervision adds value beyond the size of its dominant source, OProofs. Figure 4 holds Goedel-Prover-V2-8B, one-epoch LoRA, and direct $n = 32$ evaluation fixed. MAReason-Corpus leads on Level I, while expanding general OProofs supervision does not consistently improve over the base model. MAReason-Corpus's advantage over OProofs-Matched appears at smaller budgets; their Pass@32 is tied, suggesting more reliable generation

Table 3: Planner/filter ablation on Hard-Dev (40 problems). Fractions show successful/total counts for accepted blueprints, legal auxiliary statements, proved auxiliary obligations, and fully verified problems (Final). Hall. is H/F (%) over all failed auxiliary and final candidates. Column-best values are bold (including ties): higher success rates are better for fractions, and lower is better for Hall.

Config.	Blueprint model	Premise filter	Blueprint	Legal aux.	Proved aux.	Final	Hall.
A	OProver-8B	Qwen2.5-Math-7B-Instruct	32/40	96/101	44/90	12/40	15.207
B	GPT-5.6-sol	Qwen2.5-Math-7B-Instruct	**39/40**	142/157	**110/130**	28/40	14.497
C	OProver-8B	Gemini-3.7-Flash	35/40	125/136	32/105	9/40	**13.873**
D	GPT-5.6-sol	Gemini-3.7-Flash	**39/40**	**141/145**	108/128	**29/40**	14.950

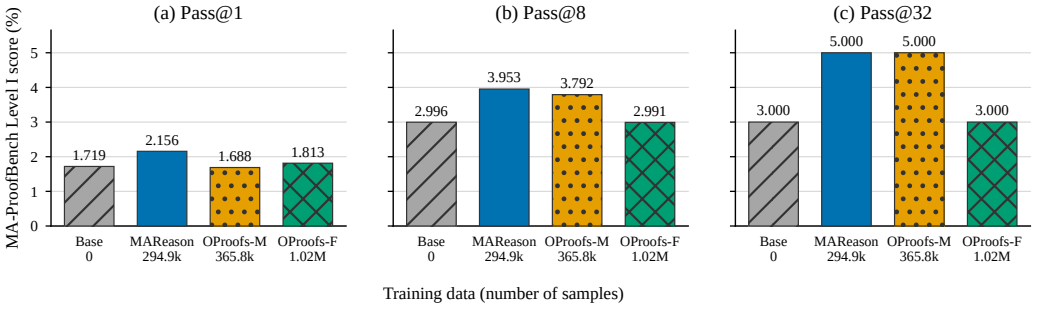


Figure 4: Training-data ablation for Goedel-Prover-V2-8B under direct MA-ProofBench Level I evaluation with $n = 32$. OProofs-M and OProofs-F denote OProofs-Matched and OProofs-Full. Numbers below the training-data labels are sample counts. All Level II scores are zero and are omitted.

within similar coverage. All conditions fail on Level II. Domain alignment thus helps easier analysis proofs, but this final-proof supervision does not recover the harder proofs targeted by structured inference.

5 Discussion

The experiments support verification-triggered escalation across proof granularities. Standard Retrieval improves whole-proof grounding, while Deep Mode extends coverage through independently grounded and verified obligations. Planning and repair also add computation; matched-compute experiments are needed to separate structural benefits from search effort. Escalation reserves Deep Mode for unresolved goals. Most benchmark problems still reach this stage, and large compute savings remain unestablished. Low Level II coverage points to a further bottleneck: even well-formed subgoals can resist local proving or final composition. Budget-aware stopping, stronger local proving, and transfer beyond analysis are natural next tests of this approach.

The data ablation also distinguishes generation reliability from eventual coverage: MAReason-Corpus outperforms OProofs-Matched at smaller sampling budgets despite tied Pass@32. This motivates combining domain adaptation with selective escalation, whose joint benefit remains to be tested.

6 Conclusion

MAReason uses Lean verification to trigger a progression from premise-augmented whole-proof generation to independently grounded and verified obligations when monolithic proving fails. On MA-ProofBench, this staged search improves proof success and reduces Mathlib hallucination among failed candidates relative to direct generation with the same fixed prover. Separately, MAReason-Corpus improves undergraduate-analysis proving through domain adaptation, while harder qualifying-exam problems remain unresolved by this supervision alone. The results support a complementary role for domain-specific supervision and inference-time proof structure, with Lean verification linking proposed reasoning to complete formal proofs.

AI USE STATEMENT

The authors conceived the core research question and method and made all final methodological and experimental decisions. Generative AI tools were used to assist with experiment design, code implementation and debugging, data-processing scripts, experiment execution, figure preparation, and manuscript drafting and editing. GPT and Gemini models were also used as explicit components of the evaluated MAReason inference pipeline, as described in Sections 3 and 4.

All reported numerical results were derived from retained experiment artifacts and independently checked against candidate-level logs. Generated Lean proofs were validated using the stated Kimina/Lean environment. The authors manually verified citations, reviewed AI-assisted code and text, and take full responsibility for the methods, claims, results, and final manuscript.

ETHICS STATEMENT

This work studies formal mathematical reasoning and does not involve human participants, personal data, or decisions about individuals. Its main research integrity risks are benchmark contamination and overstating unverified model outputs. We address these risks through explicit overlap removal, preservation of data provenance, and compiler-based validation of reported proofs. Any release of derived data or model artifacts should preserve the licenses and attribution requirements of the underlying resources.

REPRODUCIBILITY STATEMENT

Sections 3 and 4.1 specify the method, benchmark, metrics, generation protocol, and verification environment. Appendix A records the frozen inference budgets, prompts, accessibility filters, blueprint gates, repair limits, and final Kimina/Lean verification procedure. Appendix B documents corpus filtering, source composition, token coverage, and LoRA settings. We retain seeds, per-problem retrieval and planning caches, generated candidates, Lean diagnostics, and final verification artifacts so that every reported aggregate can be traced to candidate-level evidence.

REFERENCES

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023. URL https://arxiv.org/abs/2302.12433.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4WnqRR915j.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated Lean 4 dataset. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Se6MgCtRhz.

Guoxiong Gao, Zeming Sun, Jiedong Jiang, Yutong Wang, Jingda Xu, Peihao Wu, Bryan Dai, and Bin Dong. LeanSearch v2: Global premise retrieval for Lean 4 theorem proving. *arXiv preprint arXiv:2605.13137*, 2026. URL https://arxiv.org/abs/2605.13137.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025. URL https://arxiv.org/abs/2505.22312.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-Reasoner-Zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025. URL https://arxiv.org/abs/2503.24290.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/a8901c5e85fb8e1823bbf0f755053672-Abstract-Conference.html.

Joshua Ong Jun Leang, Zheng Zhao, Mihaela Catalina Stoian, Qiyuan Xu, Haonan Li, Wenda Li, Shay B. Cohen, and Eleonora Giunchiglia. Pythagoras-Prover: Advancing efficient formal proving via augmented lean formalisation. *arXiv preprint arXiv:2606.12594*, 2026. URL https://arxiv.org/abs/2606.12594.

Xingxuan Li, Yao Xiao, Dianwen Ng, Hai Ye, Yue Deng, Xiang Lin, Bin Wang, Zhanfeng Mo, Chong Zhang, Yueyi Zhang, Zonglin Yang, Ruilin Li, Lei Lei, Shihao Xu, Han Zhao, Weiling Chen, Feng Ji, and Lidong Bing. MiroMind-M1: An open-source advancement in mathematical reasoning via context-aware multi-stage policy optimization. *arXiv preprint arXiv:2507.14683*, 2025. URL https://arxiv.org/abs/2507.14683.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-STaR: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024. URL https://arxiv.org/abs/2407.10040.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025. URL https://arxiv.org/abs/2508.03613.

Jialin Lu, Kye Emond, Kaiyu Yang, Swarat Chaudhuri, Weiran Sun, and Wuyang Chen. Lean Finder: Semantic search for Mathlib that understands user intents. *arXiv preprint arXiv:2510.15940*, 2025. URL https://arxiv.org/abs/2510.15940.

David Ma, Kaijing Ma, Shawn Guo, Yunfeng Shi, Enduo Zhao, Jiajun Shi, Zhaoxiang Zhang, Gavin Cheung, Jiaheng Liu, and Zili Wang. OProver: A unified framework for agentic formal theorem proving. *arXiv preprint arXiv:2605.17283*, 2026. URL https://arxiv.org/abs/2605.17283.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2303.04488.

Job Petrovčič, David Eliecer Narvaez Denis, and Ljupčo Todorovski. Combining textual and structural information for premise selection in Lean. *arXiv preprint arXiv:2510.23637*, 2025. URL https://arxiv.org/abs/2510.23637.

Bartosz Piotrowski, Ramon Fernández Mir, and Edward Ayers. Machine-learned premise selection for Lean. *arXiv preprint arXiv:2304.00994*, 2023. URL https://arxiv.org/abs/2304.00994.

Lushi Pu, Weiming Zhang, Xinheng Xie, Zixuan Fu, Bingxiang He, Hongya Lyu, Xin Li, Jie Zhou, and Yudong Wang. MA-ProofBench: A two-tiered evaluation of LLMs for theorem proving in mathematical analysis. *arXiv preprint arXiv:2606.13782*, 2026. URL https://arxiv.org/abs/2606.13782.

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. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025. URL https://arxiv.org/abs/2504.21801.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean Copilot: Large language models as copilots for theorem proving in Lean. *arXiv preprint arXiv:2404.12534*, 2024. URL https://arxiv.org/abs/2404.12534.

TaoAnalysisBench. TaoAnalysisBench: A Lean 4 theorem-proving benchmark derived from Tao's Analysis I formalization. Hugging Face dataset, v1.0.0, 2026. URL https://huggingface.co/datasets/taoanalysisbench/TaoAnalysisBench.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. PutnamBench: Evaluating neural theorem-provers on the putnam mathematical competition. *arXiv preprint arXiv:2407.11214*, 2024. URL https://arxiv.org/abs/2407.11214.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=3f5PALef5B.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. TheoremLlama: Transforming general-purpose LLMs into Lean4 experts. *arXiv preprint arXiv:2407.03203*, 2024b. URL https://arxiv.org/abs/2407.03203.

Zijian Wu, Jiayu Wang, Dahua Lin, and Kai Chen. LEAN-GitHub: Compiling github LEAN repositories for a versatile LEAN prover. *arXiv preprint arXiv:2407.17227*, 2024. URL https://arxiv.org/abs/2407.17227.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024. URL https://arxiv.org/abs/2408.08152.

Ran Xin, Zeyu Zheng, Yanchen Nie, Kun Yuan, and Xia Xiao. Scaling up multi-turn off-policy RL and multi-agent tree search for LLM step-provers. *arXiv preprint arXiv:2509.06493*, 2025. URL https://arxiv.org/abs/2509.06493.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024. URL https://arxiv.org/abs/2409.12122.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.15626.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Zheng Yuan, Dahua Lin, and Kai Chen. Lean Workbook: A large-scale Lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*, 2024. URL https://arxiv.org/abs/2406.03847.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Yuan Zheng, Huajian Xin, Wenhao Huang, Yandong Wen, and Weiyang Liu. FormalMATH: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025. URL https://arxiv.org/abs/2505.02735.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021. URL https://arxiv.org/abs/2109.00110. Lean 4 port: https://github.com/yangky11/miniF2F-lean4.

A MAREASON REPRODUCIBILITY DETAILS

A.1 FROZEN INFERENCE PROTOCOL

Table 4 records the configuration used by the main OProver-8B system. The runtime context is 36,864 tokens: at most 4,096 prompt tokens and 32,768 generated tokens for a whole-proof or bridge attempt. All eight samples use the same cached retrieval and planning artifacts but distinct deterministic seeds derived from a global seed, problem identifier, proof stage, candidate index, repair round, and trajectory index.

Table 4: Frozen MAReason inference configuration.

Component	Configuration
Proof generator	OProver-8B, weights unchanged
Samples / Deep trajectories	8
Prover sampling	temperature 1.0, top-p 0.95
Global seed	20260812
Standard Retrieval	retrieve 50, rerank to Top-10
Reasoning Retrieval	search 30, return 30, at most 3 revisions
Blueprint	2–4 auxiliary propositions
Blueprint attempts	1 initial attempt + at most 2 revisions
Proof candidates	4 per auxiliary node and final bridge
Repair budget	at most 2 repair rounds per candidate
Maximum generated tokens	8,192 auxiliary; 32,768 bridge
Lean check timeout	180 seconds
Verification environment	Lean 4.28.0 and matched Mathlib
Final verifier	MA-ProofBench Kimina reconstruction

A.2 RETRIEVAL AND PREMISE ACCESSIBILITY

The Standard query concatenates the informal statement, when present, with the Lean declaration after removing its proof placeholder. Retrieval uses the LeanSearch v2 Mathlib 4.28 index with Qwen3-Embedding-8B and Qwen3-Reranker-8B. For every retrieved list, we remove an empty declaration name, the target name, a declaration whose normalized type exactly matches the target, and repeated module–name pairs. The remaining names are compiled in the target environment and inaccessible declarations are removed. Premises are first rendered with their full declarations; if the 4,096-token prompt budget is exceeded, the renderer falls back to signatures and then to names only.

Reasoning Retrieval follows LeanSearch v2's sketch–retrieve–filter–judge loop. In the closed-role MAReason configuration, GPT-5.6-sol with `reasoning_effort=xhigh` performs sketch generation, judgment, and revision, while Gemini-3.7-Flash performs premise filtering. The respective generation limits are 4,096 and 1,024 tokens. Successful responses are cached by request content. A missing, malformed, or failed closed-role response is not replaced by a local model.

The Hard-Dev API audit records third-party routes through iKunCode and AIHubMix, including reused successful historical responses. GPT is reported under the identifier `gpt-5.6-sol`; the

filter alias is `gemini-3.7-flash`. These are requested model identifiers: the retained logs do not independently establish upstream model identity. Malformed or failed responses and local fallbacks are excluded from the reported API-role results. Valid cached responses are reused across proof samples.

A.3 Blueprint Schema and Legality Gates

The planner receives the theorem, the names available in its original context, the target-level Reasoning Retrieval trace, and at most 24 accessible premise signatures. It returns exactly the fields `overall_strategy`, `auxiliary_lemmas`, and `final_step`. Each auxiliary entry contains an identifier `aux_i`, one Lean proposition, a list of earlier auxiliary dependencies, and the original context names occurring in that proposition. Planning uses temperature zero and an 8,192-token output budget.

Before proof generation, MAReason performs three legality gates. First, it validates the JSON shape, dependency order, context names, and absence of proof holes, proof syntax inside propositions, or the target theorem. Second, it materializes declarations from the original target context; model-written declaration scaffolding is not trusted. Third, Lean checks every standalone auxiliary statement, the bridge statement, and the complete DAG in the target context. These checks use a private sentinel to inhabit temporary declarations solely for type checking. Sentinel proofs are never carried into generated proofs or counted as solutions.

A.4 Proof Search, Repair, and Final Verification

Every auxiliary node independently runs the same 30-to-30 Reasoning Retrieval procedure, with the standalone proposition as the formal query and the local proposition plus overall strategy as its descriptive context. For each node, OProver generates four candidates. Failed candidates retain separate states; each repair prompt includes that candidate's normalized code and Lean error. After the initial attempt, at most two repair rounds are allowed. The lowest indexed successful candidate is retained.

A standalone success is insufficient. MAReason extracts its proof body, inserts the node and its dependency closure into the original theorem, and checks this partial transplant in Lean. The proof search stops if no candidate for a required node passes both checks. After all nodes succeed, the same four-candidate, two-repair procedure proves the bridge. The final program is constructed by inserting the verified node bodies as local `have` declarations and replacing the original proof placeholder with the bridge body. This assembled program is checked once more during generation and subsequently evaluated with the MA-ProofBench Kimina Lean Server, including its static precheck and statement-tampering check. Results are keyed by problem and sample identifier so interrupted runs can resume without duplicating completed work or generation requests.

Planning and retrieval artifacts are cached across proof samples, while proof generation uses independent seeds. If a Deep trajectory fails, its corresponding candidate from the preceding retrieval stage is retained in the final candidate set. Earlier all-failed batches and internal auxiliary attempts are not pooled into the eight candidates used for the routed metrics.

B MAReason-Corpus Construction Details

Table 5 gives the complete filtering ledger. Compilation is deliberately applied after inexpensive completeness, domain, and duplicate filters. The final benchmark check removes one normalized statement overlap.

The final source distribution is: OProofs 271,523 (87.601%), Goedel-Prover-V2-SFT 28,721 (9.266%), Nemotron-Math-Proofs-v1 6,937 (2.238%), NuminaMath-Lean 2,140 (0.690%), Lean-Workbook-Proofs 319 (0.103%), Formal-Math-Autoformalization 305 (0.098%), and Herald-Proofs eight (0.003%). The largest domains are real analysis (226,473), limits and continuity (22,321), differentiation (15,017), complex analysis (11,529), and probability analysis (11,161). The remaining domains are measure theory (6,201), sequences and series (5,489), integration (5,456), topological analysis (5,198), functional analysis (867), harmonic/Fourier analysis (139), and operator theory (102).

Table 5: MAReason-Corpus construction ledger.

Stage	Input	Retained	Removed / skipped
Raw audit and unified parsing	10,739,063	10,476,853	262,210
Complete-proof filtering	10,476,853	9,764,061	712,792
Analysis-domain filtering	9,764,061	2,454,398	7,309,663
Cross-source deduplication	2,454,398	580,729	1,873,669
Lean 4.28 compilation	503,773	309,954	193,819
MA-ProofBench decontamination	309,954	309,953	1

The final corpus contains 309,953 unique identifiers and statement hashes. All records have `compile_success=true`, `contains_sorry=false`, and `benchmark_overlap=false`. With the Goedel tokenizer, 98.135% of examples fit within 4,096 tokens and 99.858% fit within 8,192.

B.1 Domain LoRA Configuration

The fixed training subset contains 294,885 examples, each fitting within the 4,096-token training context; 6,008 additional examples form the fixed 2% validation split. Training uses one epoch, learning rate 10^{-4}, LoRA rank 64, scale 128, dropout 0.05, per-device batch size two, gradient accumulation four, and seed 20260816. Training uses a statement-to-proof format.

C Additional Experimental Results

C.1 Standard Retrieval Top-K Ablation

Figure 3(b–c) varies the number of premises retained by Standard Retrieval while keeping OProver-8B, the 200 MA-ProofBench problems, the $n = 8$ generation protocol, and Lean verification fixed. Top-10 provides the strongest overall coverage and the lowest compiler-evidenced hallucination fraction among failed candidates on both levels. Larger premise sets do not improve Pass@8 and have higher hallucination fractions than Top-10. We therefore freeze Top-10 for all subsequent MAReason experiments.

Hallucination is $100H/F$, where H counts failed candidates attributed to Mathlib hallucination and F includes all failures, computed separately for each level from the full-population audit. Top-100 uses a names-only fallback and is therefore not a strict rendering-controlled point. The two curves are presented alongside the direct baselines in the main text.

The ten direct $n = 32$ models that solve no MA-ProofBench problem are BFS-Prover-V2-7B, Leanabell-Prover-V2-KM, Qwen2.5-Math-7B, Qwen2.5-Math-7B-Instruct, DeepSeek-R1-Distill-Qwen-7B, Llemma-7B, InternLM2-Math-7B, DeepSeek-Prover-V1.5-RL, InternLM2.5-Step-Prover-7B, and Kimina-Prover-Preview-Distill-7B. Each run has all 6,400 candidates represented in its final verification artifact. Models whose native context is shorter than the nominal 36,864-token runtime context use their native context limit; these rows are complete local baselines but are not strict 32,768-token-generation comparisons.

The OProver LoRA controls provide a useful negative result. Direct OProver-8B obtains Level I Pass@1/Pass@8/Pass@32 of 2.250/4.282/6.000, whereas LoRA on MAReason without OProofs obtains 0.344/1.453/3.000 and LoRA on full MAReason obtains 1.563/3.236/4.000. Both variants score zero on Level II. Thus the observed benefit of MAReason-Corpus is model-dependent under the current one-epoch LoRA configuration.

C.2 Hard-Dev Selection and Role Integrity

Hard-Dev is selected deterministically from the common MAReason validation split. We rank examples by the maximum tokenized theorem–proof length under the DeepSeek and Goedel tokenizers, preserve coverage of every represented analysis domain, and cap each source at 12 examples and each domain at six. Its 40 records comprise 12 OProofs, 12 Nemotron, 12 Goedel, and four Numi-

naMath examples. A private file stores reference proofs for integrity checks; generation receives the public theorem statements without those proofs.

Configurations A and C use Qwen2.5-Math-7B-Instruct for retrieval sketches and judgments and OProver-8B for blueprints. B and D use GPT for all three roles. Thus the A-to-B and C-to-D comparisons change a reasoning/planning configuration, including its generation budgets. Gemini replaces only the relevance-filter role in C and D. The proof generator remains OProver-8B in all configurations. Local retrieval-role outputs are limited to 1,024 tokens and local blueprints to 4,096. GPT reasoning and blueprint completion budgets are 4,096 and 8,192 tokens, respectively; Gemini filtering uses 1,024. These configuration comparisons therefore do not isolate a blueprint-model effect at equal budgets.

C.3 Routing and Failure Analysis

Fourteen of the 200 targets terminate after Standard Retrieval; the remaining 186 enter Deep Mode. Successful planner and filter responses are cached, no local model substitutes for a failed closed-model role, and all eight proof samples reuse the same retrieval and planning artifacts.

The final all-candidate audit attributes 49.9% of failed Level I candidates and 35.9% of failed Level II candidates to Mathlib hallucination. Compared with Standard Retrieval, MAReason solves substantially more problems without a monotone reduction in Level I hallucination. Many remaining failures instead arise from an unproved auxiliary obligation or an unsuccessful final bridge. This distinction indicates that grounding and proof completeness remain separate bottlenecks. Appendix D gives one verified success and one representative failure.

D Verified and Failed MAReason Examples

This section traces one complete escalation using the stored artifacts for MA-ProofBench Level I problem 2. Standard Retrieval and Reasoning Retrieval both fail for all eight proof samples, whereas Deep Mode produces four verified proofs. We show representative candidate 0 at the two failed stages and trajectory 0 at the successful stage. The displayed prompts are the templates used by the implementation, instantiated with this theorem. Long retrieved declaration bodies are replaced by their exact names to keep the example readable; the unabridged prompts and outputs remain in the experiment record.

D.1 A verified escalation from retrieval to Deep Mode

Target theorem.

```
import Mathlib

theorem ma_proofbench_l1_01 {a x : R} (ha : 1 < a) (hx : 1 <= x) :
    (1 / (1 + x) - 1 / (1 + a * x)) <=
      (Real.sqrt a - 1) / (Real.sqrt a + 1) := by
  sorry
```

Stage 1: Standard Retrieval prompt. The theorem is used as a single semantic query. The ten retrieved declarations are `Real.sqrt_le_one`, `one_div_pow_lt_one_div_pow_of_lt`, `one_div_le_one_div_of_le`, `Real.sqrt_one_add_le`, `Real.sqrt_div_self'`, `Nat.cast_le_pow_sub_div_sub`, `Real.one_sub_inv_le_log_of_pos`, `Real.rpowIntegrand`$_{01}$`_le_rpow_sub_one`,

Nat.cast_le_pow_div_sub, and one_div_pow_le_one_div_pow_of_le.

Prompt S1: Standard proof generation (OProver-8B).

```
You are an expert at using Lean for formal proofs. Your task is to complete
the following Lean 4 code. You have access to the following relevant
theorems and definitions found through search.

## Lean 4 Code to Complete
{the exact target theorem above}

## Relevant Search Results
{full declarations for the Standard Top-10 results listed above}

Before producing the Lean 4 code, provide a detailed proof plan and then the
final Lean 4 code. The final code MUST begin with the exact,
character-for-character original formal statement.
```

Observed Standard failure. None of the eight candidates compiles. Representative candidate 0 reaches the following step, after which Kimina reports an unknown declaration.

```
have h80 : (a - 1) * (Real.sqrt a + 1) * x <=
    (Real.sqrt a - 1) * (1 + x) * (1 + a * x) := h6
have h81 : (a - 1) * x / ((1 + x) * (1 + a * x)) <=
    (Real.sqrt a - 1) / (Real.sqrt a + 1) := by
  rw [div_le_div_iff (by positivity) (by positivity)]
  nlinarith [h80]

Kimina/Lean 4.28 error:
line 83: Unknown identifier `div_le_div_iff`
```

Stage 2: Reasoning Retrieval prompts. Reasoning Retrieval first asks for a theorem-specific search plan, filters each query's candidates, judges coverage, and revises the plan when coverage is insufficient. These are retrieval prompts: they do not ask the closed models to write the final Lean proof.

Prompt R1: proof sketch and multi-query decomposition (GPT-5.6-sol).

```
You are a Lean 4 Mathlib retrieval planner for mathematical analysis.
Analyze the target theorem and produce natural-language search queries for
definitions, high-level bridge theorems, and the lemmas needed by each proof
step. Queries must be specific to this theorem and suitable for semantic
search over Mathlib declarations.

## Target Lean theorem
{target theorem}
## Informal statement
{informal statement, or none}
## Informal proof, when available
{informal proof, or none}

Return one JSON object with definition_queries, highlevel_queries, and steps.
Each step contains description, reasoning, and queries. Do not emit Lean code.
```

Prompt R2: query-level premise filtering (Gemini-3.7-Flash).

```
You are filtering Mathlib declarations for one retrieval query. Keep only
declarations whose type and mathematical meaning can directly support the
requested proof step.

## Target theorem
{target theorem}
## Proof step
{step description}
## Search query
{semantic query}
```

```
## Candidate declarations
{retrieved Mathlib declarations}

Return one JSON object with kept_results and reasoning. kept_results contains
zero-based indices, or is empty when none are relevant.
```

Prompt R3: retrieval coverage judge and revision trigger (GPT-5.6-sol).

```
Judge whether the retrieved Mathlib declarations cover the important steps of
the proposed Lean proof plan. Mark the result bad when major support is missing
or the plan is not specific enough to implement in Lean.

## Target theorem
{target theorem}
## Proposed plan
{proof sketch}
## Available declarations
{filtered declarations}
## Filter notes
{filter reasoning}

Return one JSON object with judgment, reasoning, and suggestions_if_bad.
judgment must be good or bad. A bad judgment triggers a revised search plan.
```

For this problem, the resulting global premise set contains 30 declarations, including `inv_sub_inv`, `div_sub_div`, `div_le_div_iff`$_0$, `div_le_iff`$_0$, `Real.sq_sqrt`, `Real.sqrt_lt_sqrt`, `mul_nonneg`, and `mul_pos`. OProver then receives the same proof prompt as in Stage 1, with this 30-premise set substituted for the Standard Top-10 set.

Prompt R4: Reasoning-Retrieval proof generation (OProver-8B).

```
You are an expert at using Lean for formal proofs. Complete the target theorem
using the relevant theorems and definitions found through Reasoning Retrieval.

## Lean 4 Code to Complete
{the exact target theorem}
## Relevant Search Results
{full declarations for the filtered 30-premise global set}

Provide a detailed proof plan followed by complete Lean 4 code. Preserve the
original theorem statement character for character.
```

Observed Reasoning Retrieval failure. All eight candidates still fail compilation. Candidate 0 uses a relevant algebraic route but calls the unavailable identifier `le_div_iff`.

```
have h6 : (Real.sqrt a - 1) * (1 + x) * (1 + a * x) >=
    (a - 1) * (Real.sqrt a + 1) * x := h_main_ineq4
have h7 : (Real.sqrt a - 1) / (Real.sqrt a + 1) >=
    (a - 1) * x / ((1 + x) * (1 + a * x)) := by
  have h8 : ... >= 0 := by
    field_simp [h1.ne', h2.ne', h4.ne'] at h6 |-
    rw [le_div_iff (by positivity)]
    nlinarith [h6]

Kimina/Lean 4.28 error:
line 89: Unknown identifier `le_div_iff`
```

Stage 3: Deep Mode blueprint prompt. After both direct stages fail, the planner receives the immutable target, the Reasoning Retrieval trace, and accessible premise declarations. Its output is strict JSON and each proposed auxiliary statement is checked with a temporary `by sorry` declaration before proof generation.

Prompt D1: constrained Proof Blueprint (GPT-5.6-sol).

```
Design a formally checkable Lean 4 proof blueprint for this theorem.

Target theorem (immutable):
{target theorem}
Original context names that may be referenced: ["a", "x", "ha", "hx"]
LeanSearch Reasoning trace:
{the accepted multi-query retrieval trace}
Accessible Mathlib declarations:
{filtered accessible declarations}

Return exactly one JSON object with overall_strategy, auxiliary_lemmas,
dependencies, original_hypotheses, and final_step. Propose 2--4 auxiliary
lemmas in dependency order. Do not modify the target, add assumptions, use the
target theorem itself, or include sorry/admit. Each lemma must be strictly
simpler than the target and valid in its declared context.
```

Accepted blueprint. The planner introduces the following four Lean-legal obligations; the third depends on the second.

```
aux_1: 1/(1+x) - 1/(1+a*x) = ((a-1)*x)/((1+x)*(1+a*x))
aux_2: 0 <= (a-1)*(x-1)*(a*x-1)
aux_3: ((a-1)*x)/((1+x)*(1+a*x)) <= (a-1)/(2*(1+a))
       dependencies: [aux_2]
aux_4: (a-1)/(2*(1+a)) <= (Real.sqrt a-1)/(Real.sqrt a+1)

final_step: rewrite by aux_1; chain aux_3 and aux_4.
```

Auxiliary proof and repair prompts. Each auxiliary lemma is independently grounded by local retrieval and passed to OProver. A failed compilation is returned verbatim in the repair prompt. In trajectory 0, `aux_3` succeeds after one repair round and `aux_4` after two.

Prompt D2: one auxiliary lemma proof (OProver-8B).

```
You are an expert at using Lean for formal proofs. Complete this auxiliary
theorem using the retrieved declarations.

## Lean 4 Code to Complete
theorem ma_deep_2_aux_3 {a x : R} (ha : 1 < a) (hx : 1 <= x)
    (h_deep_2 : 0 <= (a-1)*(x-1)*(a*x-1)) :
    ((a-1)*x)/((1+x)*(1+a*x)) <= (a-1)/(2*(1+a)) := by
  sorry

## Relevant Search Results
{locally retrieved and filtered declarations for aux_3}

Return the complete theorem and preserve its statement exactly.
```

Prompt D3: Lean-feedback local repair (OProver-8B).

```
You are repairing a Lean 4 proof under Mathlib 4.28.0.

The theorem statement is correct and must remain character-for-character
unchanged:
{auxiliary theorem statement}

Previous proof attempt:
{failed candidate}

Lean compiler error:
{exact Kimina/Lean error; for aux_3, linarith failed at line 24}

Accessible retrieved declarations:
```

```
{the local premise set}

Fix only the proof. Never use sorry or admit. Do not invent theorem names.
Return the complete corrected Lean code, beginning with the exact statement.
```

Final bridge prompt and verified result. Only verified auxiliary facts are exposed to the bridge prover.

Prompt D4: final bridge (OProver-8B).

```
Complete the original theorem using the four verified facts h_deep_1,
h_deep_2, h_deep_3, and h_deep_4. Preserve the original theorem statement;
do not use sorry/admit; return complete Lean 4 code.

Recommended composition:
calc
  1/(1+x) - 1/(1+a*x)
      = ((a-1)*x)/((1+x)*(1+a*x)) := h_deep_1
  _ <= (a-1)/(2*(1+a)) := h_deep_3
  _ <= (Real.sqrt a-1)/(Real.sqrt a+1) := h_deep_4
```

The accepted assembled proof has 235 lines. Its final verified composition is:

```
have h_main : ((a - 1) * x) / ((1 + x) * (1 + a * x)) <=
    (Real.sqrt a - 1) / (Real.sqrt a + 1) := by
  have h1 := h_deep_3
  have h2 := h_deep_4
  linarith

have h_final : 1 / (1 + x) - 1 / (1 + a * x) <=
    (Real.sqrt a - 1) / (Real.sqrt a + 1) := by
  rw [h_deep_1]
  exact h_main

exact h_final

Verification: success (Kimina, Lean 4.28.0 / matched Mathlib)
```

This trace isolates the contribution of Deep Mode: broader retrieval alone finds useful division and square-root declarations but still leaves a monolithic proof vulnerable to one unavailable Mathlib declaration. The blueprint converts the target into four independently checked obligations, and compiler feedback repairs two of them before the final composition is attempted.

Representative failure. For Level I problem 37, the target proves a Lipschitz bound for an infimal convolution with values in `ENNReal`. The blueprint identifies the appropriate high-level route: prove finiteness, compare the two infima using the triangle inequality, and transfer the inequalities through `ENNReal.toReal`. Nevertheless, OProver's auxiliary attempts repeatedly misuse or invent Mathlib facts for extended nonnegative reals while handling finiteness and `sInf`; no complete auxiliary DAG and final bridge pass Lean verification. MAReason therefore retains the preceding retrieval-stage candidate, which also fails. The case shows that a mathematically sensible plan does not guarantee library-correct realization, particularly for specialized order and coercion interfaces.