ProofEvolve: Neuro-Symbolic Evolution for Formal Automated Theorem Proving

Wenqian Ye[1,2,*], **Ziwei Guan**[2,*], **Eric Xie**[1], **Bohan Liu**[1], **Shivani Modi**[2], **Buyun Zhang**[2], **Ellie Dingqiao Wen**[2], **Henry Kautz**[1], **Aidong Zhang**[1]

[1]University of Virginia, [2]Meta AI
[*]Core contributors

Automated theorem proving offers a natural foundation for recursive self-improvement in scientific discovery. However, existing neural provers do not fully preserve this recursive structure, where the learning process should be self-improving over time. Existing methods either embed proof experience into model parameters through expensive weight updates, or keep verified intermediate deductions only within the current problem. In addition, these methods also heavily rely on sparse whole-proof feedback, even when unsuccessful partial attempts contain useful discoveries. To close the gap, we propose **ProofEvolve**, a *neuro-symbolic* framework that evolves explicit, formally verified *symbolic* proof structures with *neural* models to decisively expand the knowledge boundary. In this framework, the neural model proposes variation operators, including decompositions, repairs, and schema recombinations. The symbolic Lean kernel verifies every proof transition. Over the evolution loops, ProofEvolve computes verified closure over the resulting proof directed acyclic graphs (DAGs). Within each problem, ProofEvolve evolves partial AND-OR proof DAGs in a behaviorally indexed archive. Across problems, kernel-checked schema extraction adds newly proved sub-DAGs to a persistent schema library. Proof DAGs inherit the solved results through typed schema recombination, with every residual premise exposed as a new subgoal. This evolutionary process preserves verified results from incomplete attempts and makes them available for later proofs without weakening formal soundness. Across three competition-level Lean benchmarks, ProofEvolve achieves the highest average solve rate among the evaluated proof systems.

Date: August 10, 2026
Correspondence: Wenqian Ye and Aidong Zhang at {wenqian, aidong}@virginia.edu

Meta

1 Introduction

Theorem proving paves the way for scientific reasoning to discover new knowledge with formal guarantee of correctness. A proof often contributes more than its stated conclusion. It can produce reusable lemmas, reveal hidden structures, and expose new hypotheses. Even unsuccessful proof programs can push forward important advances. For centuries, mathematicians attempted to derive *Euclid's parallel postulate* from his remaining axioms. Examining geometries in which the postulate does not hold instead led to the development of non-Euclidean geometry (Bonola, 1955). A similar pattern is in theoretical computer science (TCS). Hilbert's *Entscheidungsproblem* asked whether a general procedure could determine the validity of any statement in first-order logic (Hilbert and Ackermann, 1928). Following the unsolved problem, Church and Turing proved that no such general procedure exists (Church, 1936; Turing, 1937). Turing's analysis also introduced a formal model of computation that later became the foundation of TCS. Therefore, proofs fundamentally convert individual results into reusable knowledge that supports later discoveries.

Recently, AI systems have begun to automate parts of scientific discovery. Trained neural models with automatic evaluation have produced new algorithms, mathematical constructions, and structural patterns (Romera-Paredes et al., 2024; Novikov et al., 2025; Davies et al., 2021). Evolutionary search is also promising as it can improve candidates over time through repeated generation, evaluation, and selection. However, current linguistic-based neural theorem provers still heavily use a small part of the verified structure. Training-based systems store experience mainly in model parameters, so new results affect later problems only after another

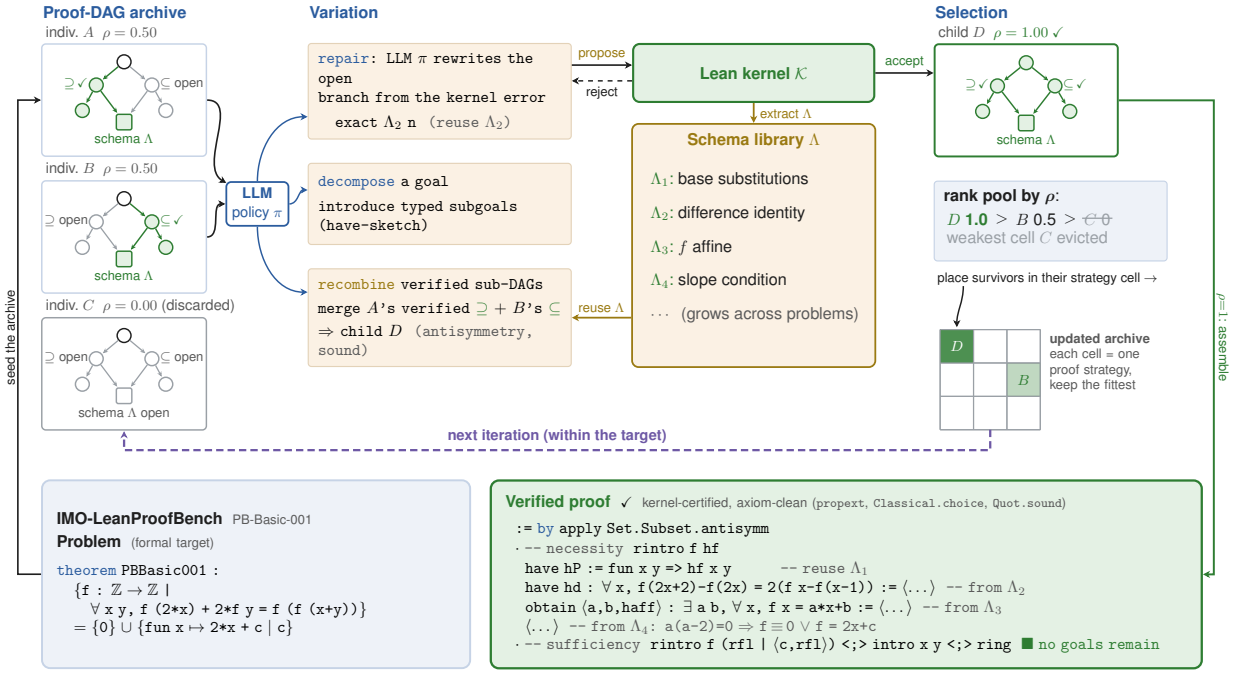


Figure 1 Overview of ProofEvolve. Within a target, a proof DAG archive preserves structurally diverse candidates and ranks them by verified closure ρ, computed from kernel-accepted subgoals. Across targets, checked extraction adds newly proved results to a persistent schema library, and typed recombination instantiates them in later DAGs while exposing all residual premises as subgoals.

training cycle (Hubert et al., 2026; Ren et al., 2025; Lin et al., 2025b). Agentic systems leverage multi-agent collaboration to work on problems with subgoals during inference (Jiang et al., 2023; Varambally et al., 2025). One latest work, LEAP (Kung et al., 2026), also shares intermediate lemmas across branches of a proof DAG, but this memory remains tied to the current target. Thus, current systems still focus mainly on whether the root theorem is solved.

Formal theorem proving in Lean 4 (de Moura and Ullrich, 2021) provides a reliable setting for cumulative evolution because every accepted proof step is rigorously verified. However, current neural theorem provers do not fully preserve verified progress. Training based methods store experience mainly in model parameters (Hubert et al., 2026; Ren et al., 2025; Lin et al., 2025b), while agentic methods reuse information mainly within the current problem (Jiang et al., 2023; Varambally et al., 2025; Kung et al., 2026). As a result, useful structures from partial and completed proofs are rarely inherited across searches. This is a central limitation for evolutionary search, which requires useful structures to survive unsuccessful candidates and pass to later generations. A small textual mutation may invalidate a complete proof even when much of its verified argument remains correct (Nagashima, 2019). A self-improving prover should therefore evolve verified partial structures rather than complete proof texts alone and preserve them for recombination across problems.

To address these shortcomings, we propose **ProofEvolve**, a neuro-symbolic evolutionary framework in which improvement grows in an explicit body of verified symbolic structures with neural proposals. As shown in Figure 1, ProofEvolve grows a proof DAG for each theorem through kernel-verified transitions. At each step the language model proposes a decomposition, repair, or schema recombination, and Lean 4 kernel accepts or rejects it. A behaviorally indexed archive (Mouret and Clune, 2015) keeps structurally diverse partial proofs and ranks them by *verified closure*, a kernel-grounded score that aggregates proved subgoals through the AND-OR structure, so selection acts on graded progress rather than a single pass-or-fail verdict. Across problems, ProofEvolve extracts closed sub-DAGs as theorem schemas. Typed recombination later instantiates a schema at a matching goal, exposes any remaining premises as new subgoals, and re-checks the result with the kernel. Verified results from earlier problems therefore enlarge the reachable search space of later ones.

To show the effectiveness, we evaluate our framework on three competition-level benchmarks, PutnamBench,

IMO-LeanProofBench, and CombiBench, with a frontier LLM (Claude Opus 4.8) as the base model under matched per-target budgets, and on a disjoint Lean Workbook split with the frontier open-weight model Qwen3.5-397B-A17B-FP8. ProofEvolve achieves an average solve rate of 57.8%, compared with 50.5% for LEAP and 45.9% for Hilbert. Ablation and test-time scaling studies further analyze the method's components and its behavior as the per-target budget grows. A separate evaluation isolates the library itself: on 744 Lean Workbook theorems disjoint from the 9,968-theorem source stream, 5,546 of the prover's own kernel-checked proofs improve the solve rate by about four points over zero-shot, whereas random retrieval from the same library gives no improvement.

In summary, our main contributions are as follows:

- We formulate theorem proving as an evolution process over structured symbolic knowledge with persistent inheritance through a Lean-verified theorem schema library, where the neural models propose new directions to extend the knowledge boundary.
- We introduce verified closure as a kernel-grounded fitness measurement and typed schema recombination as a mechanism to enable verified subproofs across targets.
- Extensive experiments on ProofEvolve against state-of-the-art neural models and agentic baselines on three competition-level Lean benchmarks show the significant effectiveness of the proposed framework, and on 744 Lean Workbook theorems disjoint from its library the prover's own verified proofs add about four points over zero-shot, while random retrieval from the same library adds nothing.

2 Related Work

2.1 Neural theorem provers

Neural theorem provers train an LLM policy to propose tactics or complete proofs in a formal language. GPT-f (Polu and Sutskever, 2020) and PACT (Han et al., 2021) established language-model policies for tactic generation. Other works (Yang et al., 2023; Lample et al., 2022; Xin et al., 2025a,b) combine learned proposals with retrieval or tree search. Recent systems obtain stronger policies from synthetic proofs, supervised fine-tuning, and reinforcement learning (Ren et al., 2025; Lin et al., 2025b; Wang et al., 2025; ByteDance Seed, 2025; Ji et al., 2025). AlphaProof (Hubert et al., 2026) trains at scale on kernel-checked self-generated experience and performs test-time adaptation on difficult targets. Retrieval-augmented provers reuse existing library lemmas at inference time (Shen et al., 2025), while self-play systems learn from both successful and failed proof trees (Poesia et al., 2024). ProofEvolve instead keeps the model fixed and stores newly proved results as explicit, verified schemas.

2.2 Agentic theorem provers

Agentic systems use multiple language models to structure proof search through decomposition, retrieval, and verifier feedback. Draft-Sketch-Prove (Jiang et al., 2023) turns informal arguments into formal sketches. COPRA (Thakur et al., 2024) constructs proofs through repeated tactic execution. Hilbert (Varambally et al., 2025) recursively decomposes difficult goals and repairs failed proofs, while LEAP (Kung et al., 2026) uses an AND-OR proof DAG to share intermediate lemmas across branches. AlphaProof Nexus (Tsoukalas et al., 2026) applies compiler-guided agents to research problems, while AlphaGeometry (Trinh et al., 2024) combines neural proposals with symbolic deduction in geometry. These methods strengthen search within a target. In contrast, ProofEvolve additionally extracts verified sub-DAGs for reuse across targets, which enables the framework the ability to recursively evolve.

2.3 Symbolic knowledge evolution

Evolutionary program search, including FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025), alternates language model generation with automatic evaluation and selection, while MAP-Elites preserves diverse high quality candidates across behavioral niches (Mouret and Clune, 2015). LEGO-Prover expands a lemma library (Wang et al., 2023), although storage alone does not guarantee reuse (Berlot-Attwell et al., 2025). DreamProver is closest to our setting because it keeps the model fixed and builds a transferable

Lean library through wake sleep abstraction (Zhang et al., 2026; Ellis et al., 2021). Other methods retrain a retriever (Kumarappan et al., 2024), distill proof strategies (Fang et al., 2025), or modify agent code using empirical fitness (Zhang et al., 2025). Formal proof evolution is difficult because verification is binary and small edits can invalidate useful candidates (Nagashima, 2019). ProofEvolve instead combines verified closure and typed schema recombination in a single inference time process. Partial proof DAGs are selected through kernel accepted subgoals, while newly closed sub-DAGs become schemas reused in later candidates without modifying the model. Under Kautz's taxonomy on Neural Symbolic AI (Kautz, 2022), ProofEvolve is a NEURO[SYMBOLIC] system where the Lean kernel and verified closure operator are embedded in the neural generation process to recursively improve over time.

3 Preliminaries

We first define the symbolic structures on which ProofEvolve operates. Every state is a Lean 4 artifact, and the search retains each partial result rather than discarding it. A closed fragment of an attempt is a valid, reusable result even before the attempt closes its root goal (Eq. (5)). This property is what later lets the search accumulate and transfer verified work (Section 4).

Lean verification. We work in a fixed Lean 4 environment $\mathcal{E}$ that imports a matched Mathlib version (The mathlib Community, 2020). We write $\mathcal{E};\Gamma \vdash_{\mathcal{K}} p : g$ when the term p elaborates under local context Γ. It contains no unresolved metavariables or placeholders, and is accepted at type g by Lean's kernel $\mathcal{K}$. A tactic state is $s = (\Gamma \vdash g)$ (Polu and Sutskever, 2020; Yang et al., 2023), and its checked witnesses form

$$\mathsf{Prf}_{\mathcal{E}}(s) = \{p \mid \mathcal{E};\Gamma \vdash_{\mathcal{K}} p : g\}. \tag{1}$$

We write $u \equiv_{\mathcal{E}} v$ for definitional equality in $\mathcal{E}$, let $\mathsf{Thm}(\mathcal{E})$ denote the theorem declarations imported into $\mathcal{E}$, and call a target T *well-formed* when $\mathcal{E};\varnothing \vdash_{\mathcal{K}} T : \mathsf{Prop}$.

Reusable proof structures. For a *well-formed* target T, a proof attempt is a finite acyclic AND-OR proof DAG $D = (V, E, r)$ with root $r = (\Gamma_0 \vdash T)$, where each node is a tactic state. An accepted hyperedge $e = (s; s_1, \ldots, s_k)$ is one Lean-elaborated proof constructor from the child obligations to the source obligation, i.e., a checked realizer

$$F_e : \prod_{i=1}^{k} \mathsf{Prf}_{\mathcal{E}}(s_i) \longrightarrow \mathsf{Prf}_{\mathcal{E}}(s). \tag{2}$$

For $k = 0$, the empty product is the singleton $\{\star\}$ and $F_e(\star) \in \mathsf{Prf}_{\mathcal{E}}(s)$ is a checked closing witness. Multiple edges leaving s are alternative proof steps, whereas the children of a single edge must all be discharged. For DAGs sharing a root, we write $D \preceq D'$ when $V(D) \subseteq V(D')$, $E(D) \subseteq E(D')$, and all existing node labels and edge realizers are preserved, and we call D' an *extension* of D.

Let $\mathrm{out}_D(s)$ be the accepted edges leaving s and $\mathrm{ch}(e)$ be their children. Since D is acyclic, closure is well defined by

$$\mathrm{Closed}_D(s) \iff \exists e \in \mathrm{out}_D(s)\ \forall s' \in \mathrm{ch}(e),\ \mathrm{Closed}_D(s'). \tag{3}$$

The universal condition is vacuous for a checked closing edge. For each closed s, we fix one witnessing edge $\mathrm{win}_D(s)$, and the constructions below hold for any such choice. The open search boundary is

$$\mathrm{frontier}(D) = \{s \in V : \neg\mathrm{Closed}_D(s),\ \mathrm{out}_D(s) = \varnothing\}. \tag{4}$$

Writing $\mathrm{win}_D(s) = (s; s_1, \ldots, s_k)$ and taking the empty tuple to be $\star$, proof assembly is the recursion

$$\mathrm{Asm}_D(s) = F_{\mathrm{win}_D(s)}\left((\mathrm{Asm}_D(s_i))_{i=1}^{k}\right). \tag{5}$$

Acyclicity makes this recursion well founded, and Eq. (2) gives $\mathrm{Asm}_D(s) \in \mathsf{Prf}_{\mathcal{E}}(s)$: a closed internal sub-DAG is a typed result, reusable while its root remains open.

4 Neuro-Symbolic Evolution for Formal Automated Theorem Proving

4.1 Overview

Figure 1 shows the ProofEvolve pipeline. A neural language model proposes structural variations, the Lean kernel checks every proof transition and schema application, and the search keeps the checked structures that make the most verified progress. *Verified closure* (Sec. 4.2) turns the kernel's binary verdict into a graded fitness read off the proof DAG, so two unfinished attempts can still be ranked. *Schema recombination* (Sec. 4.3) carries a verified result from one proof into another as a single kernel-checked step. Within one target, a *proof DAG archive* stores candidate proofs; across targets, a *schema library* stores proved results. Schema extraction adds results to the library, and schema recombination applies them in later DAGs.

ProofEvolve separates neural proposal from symbolic state transition. The policy π proposes edits and the semantic retriever supplies premises and schemas, but every change to the proof state passes through the symbolic operators defined below. At evolutionary iteration t, let $\mathcal{Q}_t$ be the target queue, $\mathcal{M}_{T,t}$ the proof DAG archive for target T, $\mathcal{L}_t$ the verified theorem schema library, and $\mathcal{H}_t$ the store of rejected proposals and Lean errors keyed by (T, D, s). The operational state is

$$\mathcal{S}_t = (\mathcal{Q}_t, \{\mathcal{M}_{T,t}\}_{T\in\mathcal{Q}_t}, \mathcal{L}_t, \mathcal{H}_t)\,. \tag{6}$$

Each $\mathcal{M}_{T,t}$ is local to one target, whereas $\mathcal{L}_t$ persists across targets. The trusted projection is $\Sigma_t = (\{\mathcal{M}_{T,t}\}_T, \mathcal{L}_t)$, and only kernel-verified DAG transitions, archive updates with accepted DAGs, and kernel-checked schema insertions may modify it.

4.2 Neural Proof Proposal

At each step the policy π proposes an edit δ to one frontier state $s \in \text{frontier}(D)$, and the kernel decides whether it survives. The trusted transition operator is

$$\text{step}_{\mathcal{K}}(D, \delta) = \begin{cases} D', & \begin{array}{c}\text{if Lean accepts the induced edge and}\\ D' \text{ is an acyclic extension of } D\end{array}\\ \bot, & \text{otherwise.}\end{cases}, \tag{7}$$

An accepted D' extends D by a typed edge carrying the realizer of Eq. (2) and stays finite and acyclic, whereas a rejected proposal leaves the proof DAG archive and schema library unchanged. A proposal takes one of three forms. In *decomposition*, the model proposes either a checked closing term or a proof constructor with typed intermediate obligations, and any proposal-time hole must become an explicit child state before acceptance (Jiang et al., 2023; Varambally et al., 2025). In *schema recombination*, the model applies a library schema, whose mechanics we defer to Section 4.3. In *repair*, available only for a previously rejected proposal at the same state, the model receives the proposal, retrieval context, and Lean error, and the corrected edge must pass the same $\text{step}_{\mathcal{K}}$ check (Varambally et al., 2025).

Kernel-grounded selection. Root verification is binary, but an accepted DAG records which internal obligations are already proved, and we turn this structure into a kernel-grounded fitness functional. For every nonempty child set, let $w_e(s') > 0$ with $\sum_{s'\in\text{ch}(e)} w_e(s') = 1$. Edge and state values are defined together by well-founded recursion over the acyclic DAG, evaluated from leaves to root in reverse topological order. For a non-closing edge,

$$\rho_D(e) = \sum_{s'\in\text{ch}(e)} w_e(s')\rho_D(s'). \tag{8}$$

For a state s,

$$\rho_D(s) = \begin{cases} 1, & \text{Closed}_D(s),\\ 0, & \text{out}_D(s) = \varnothing,\\ \max\limits_{e\in\text{out}_D(s)} \rho_D(e), & \text{otherwise,}\end{cases} \tag{9}$$

We use uniform weights and write $\rho(D) = \rho_D(r)$. The maximum encodes alternative edges at an OR node, and the weighted sum encodes the conjunctive obligations along an AND edge. Reverse topological induction

gives $\rho(D) \in [0,1]$. If r is closed, the first case of Eq. (9) gives $\rho(D) = 1$. Conversely, if an open state has value one, some outgoing edge has weighted average one, and positivity of the weights forces every child to have value one. Induction then closes every child and hence the source, a contradiction. Therefore

$$\rho(D) = 1 \iff \mathrm{Closed}_D(r). \tag{10}$$

Finally, if $D \preceq D'$, every earlier alternative remains available and closed nodes stay closed, so $\rho(D') \geq \rho(D)$: verified closure is monotone under extension and computed entirely from kernel-accepted proof DAGs.

Structural diversity. To preserve distinct proof strategies, let $\mathcal{B}$ be a fixed descriptor space and let $b(D) \in \mathcal{B}$ record binned depth, dominant tactic family, and the region of the schema index used by D. The archive $\mathcal{M}_T$ (Mouret and Clune, 2015) stores at most one DAG per descriptor, and an accepted challenger D' updates its cell by

$$\mathcal{M}_T[b(D')] \leftarrow \begin{cases} D', & b(D') \notin \mathrm{dom}(\mathcal{M}_T), \\ D', & \rho(D') > \rho(\mathcal{M}_T[b(D')]), \\ \mathcal{M}_T[b(D')], & \text{otherwise}, \end{cases} \tag{11}$$

so the incumbent wins ties. For a temperature $\tau > 0$, parents are sampled from occupied cells by

$$P(D \mid \mathcal{M}_T) = \frac{\exp(\rho(D)/\tau)}{\sum_{D'' \in \mathrm{range}(\mathcal{M}_T)} \exp(\rho(D'')/\tau)}. \tag{12}$$

The archive therefore retains structural diversity while verified closure supplies selection pressure.

Frontier scheduling. Algorithm 1 (Appendix A) gives one iteration of ProofEvolve, with parent selection following Eq. (12). For $s \in \mathrm{frontier}(D)$, let $\rho_D^{[s \mapsto 1]}$ denote the recursion of Eqs. (8)–(9) with the value at s counterfactually fixed to one. Frontier scheduling then uses

$$\Delta_D(s) = \rho_D^{[s \mapsto 1]}(r) - \rho_D(r), \tag{13}$$

which prioritizes the frontier state whose closure would yield the largest structural gain.

4.3 Symbolic Knowledge Inheritance

Recombination is the operator that carries verified work across targets. A closed state may depend on variables and assumptions from its local context, and kernel-checked schema extraction turns that local result into a reusable pair (ℓ, π_ℓ): the schema ℓ quantifies the free local variables $\mathbf{x}$, makes each used hypothesis A_i an explicit premise, and has conclusion C, while π_ℓ is its proof term:

$$\ell : \forall \mathbf{x},\ A_1 \to \cdots \to A_m \to C, \qquad \mathcal{E} \vdash_{\mathcal{K}} \pi_\ell : \ell. \tag{14}$$

For this prenex schema, $\mathrm{concl}(\ell) = C$. Writing $\mathrm{Cl}(D) = \{s \in V(D) : \mathrm{Closed}_D(s)\}$, the states newly closed by an extension $D \preceq D'$ are

$$\mathrm{NewClose}(D, D') = \mathrm{Cl}(D') \setminus \mathrm{Cl}(D). \tag{15}$$

For each $s \in \mathrm{NewClose}(D, D')$, the extraction map follows $\mathrm{win}_{D'}$ and abstracts, in dependency order, exactly the free local constants and hypotheses occurring in $\mathrm{Asm}_{D'}(s)$. It returns no schema if this generalization or the displayed kernel judgment fails. We write $\mathrm{Extract}_{\mathcal{K}}(D, D')$ for the checked schemas obtained from this set.

At an open state $s = (\Gamma \vdash g)$, a schema is applicable when a typed substitution makes its conclusion definitionally equal to the goal. A substitution σ maps the binder telescope $\mathbf{x}$ to Lean terms in Γ, and we write $\mathrm{Adm}_\Gamma(\sigma)$ when every instantiated binder elaborates in Γ without unresolved metavariables. Then

$$\mathsf{App}_{\mathcal{L}}(s) = \{(\ell, \sigma) : \mathrm{concl}(\ell)\sigma \equiv_{\mathcal{E}} g \wedge \mathrm{Adm}_\Gamma(\sigma)\}. \tag{16}$$

For $(\ell, \sigma) \in \mathsf{App}_{\mathcal{L}}(s)$, Lean first attempts local witnesses $p_i \in \mathsf{Prf}_{\mathcal{E}}(\Gamma \vdash A_i\sigma)$. Each premise without such a witness becomes a child state $(\Gamma \vdash A_i\sigma)$, and the instantiated schema application is elaborated as a complete

hyperedge realizer before acceptance. The semantic retriever returns a Mathlib premise shortlist and a library-schema shortlist,

$$\mathcal{R}_M(s) \subseteq \mathsf{Thm}(\mathcal{E}), \qquad \mathcal{R}_{\mathcal{L}}(s) \subseteq \{\ell \mid (\ell, \pi_\ell) \in \mathcal{L}\}, \tag{17}$$

and the candidates presented to the policy are

$$\mathcal{C}_{\mathcal{L}}(s) = \{(\ell, \sigma) \in \mathsf{App}_{\mathcal{L}}(s) : \ell \in \mathcal{R}_{\mathcal{L}}(s)\}. \tag{18}$$

To recombine, the model selects $(\ell, \sigma) \in \mathcal{C}_{\mathcal{L}}(s)$, and Lean accepts the result after checking the instantiated schema, all local witnesses, and all residual child obligations as one realizer. Mathlib retrieval separately supplies premises to the proposal context. A persistent schema therefore enters a later proof only as a kernel-checked transformation of its state.

4.4 Theoretical analysis

We formalize the soundness of ProofEvolve by showing that every proof object accepted into the proof DAG archive or schema library is validated by the Lean kernel. The result below states that this property is preserved as the search extends proof DAGs and grows the library. Let $\mathfrak{D}_0$ contain the initial DAGs. For $t > 0$, $\mathfrak{D}_t$ also contains every challenger accepted during transitions $0, \ldots, t-1$, including challengers later discarded by archive comparison.

Theorem 1 (Invariance of kernel-grounded state)**.** *Under the formal assumptions in Appendix B, every finite execution $\mathcal{S}_0 \to \cdots \to \mathcal{S}_n$ of Algorithm 1 satisfies, for each $0 \le t \le n$:*

(I1) *every accepted edge in every $D \in \mathfrak{D}_t$ has a checked realizer of the form in Eq. (2);*
(I2) *every closed node s in every $D \in \mathfrak{D}_t$ satisfies $\mathrm{Asm}_D(s) \in \mathsf{Prf}_{\mathcal{E}}(s)$; and*
(I3) *every $(\ell, \pi_\ell) \in \mathcal{L}_t$ satisfies $\mathcal{E} \vdash_{\mathcal{K}} \pi_\ell : \ell$.*

For every $0 \le t < n$, $\mathcal{L}_t \subseteq \mathcal{L}_{t+1}$. If transition t rejects its proposal, then $\Sigma_{t+1} = \Sigma_t$.

Building on Theorem 1, we obtain the validity of the proofs ProofEvolve returns.

Corollary 1 (Validity of returned proofs)**.** *Under the assumptions of Theorem 1, if ProofEvolve returns p from $D' \in \mathfrak{D}_n$ for a target T with root $r = (\Gamma_0 \vdash T)$, then $\mathcal{E}; \Gamma_0 \vdash_{\mathcal{K}} p : T$.*

The neural models decide only which variations to attempt, whereas the kernel decides whether an accepted one is valid. Every proof ProofEvolve returns therefore type-checks against the standard axioms by construction. The assumptions and full proofs of Theorem 1 and Corollary 1 are deferred to Appendix B.

5 Experiments

5.1 Experimental setup

Implementation. In our experiments, the parameters of all base LLMs remain frozen throughout search and across targets. For all agentic baselines, we use Claude Opus 4.8 as base model to ensure fair comparison. Lean 4 with Mathlib provides tactic states, elaboration errors, and kernel verification. Each target receives a budget of parallel attempts together with a bounded kernel-guided repair loop, and ProofEvolve draws its attempts from this budget. All experiments use Lean 4 with the same Mathlib version. For proprietary models, we use API calls. For open-sourced model hosting, we use NVIDIA B200 GPU clusters with 192 GB of memory per GPU. Each node contains 8 GPUs, and our largest runs use up to 28 nodes, corresponding to 224 GPUs operating concurrently.

Benchmarks and baselines. We evaluate on three competition-level benchmarks. PutnamBench (Tsoukalas et al., 2024) formalizes problems from the William Lowell Putnam Mathematical Competition. We use its pure-proof subset, which excludes problems whose theorem statements already contain a fixed answer value. IMO-LeanProofBench (Luong et al., 2025) contains Lean formalizations of International Mathematical Olympiad-level proof problems. CombiBench (Liu et al., 2025) covers competition-level combinatorial mathematics,

Method	Putnam (%)	IMO-Lean (%)	Combi (%)	Avg. (%)
Inference-only models				
Claude Haiku 4.5 (Anthropic, 2025)	0.0	0.0	3.3	1.1
Claude Sonnet 4.6 (Anthropic, 2026b)	0.0	0.0	6.7	2.2
Claude Opus 4.8 (Anthropic, 2026a)	0.0	0.0	10.0	3.3
Gemini 3.1 Pro (Google DeepMind, 2026)	0.0	3.3	10.0	4.4
DeepSeek-Prover-V2-671B (Ren et al., 2025)	7.0	0.0	10.0	5.7
Goedel-Prover-V2-32B (Lin et al., 2025b)	12.8	5.0	0.0	5.9
GPT-5.5 (OpenAI, 2026)	10.0	5.0	13.0	9.3
Agentic systems				
ReAct (Yao et al., 2023) (Claude Opus 4.8)	35.0	15.0	27.0	25.7
Aristotle (Achim et al., 2025) (Claude Opus 4.8)	45.0	13.3	40.0	32.8
AxProver (Breen et al., 2025) (Claude Opus 4.8)	54.3	10.0	47.0	37.1
Hilbert (Varambally et al., 2025) (Claude Opus 4.8)	55.5	33.3	49.0	45.9
LEAP (Kung et al., 2026) (Claude Opus 4.8)	64.7	36.7	**50.0**	50.5
Our work				
ProofEvolve (Claude Opus 4.8)	**71.2**	**53.3**	49.0	**57.8**

Table 1 Main comparison. Mean solve rate (%) over three independent runs on PutnamBench (Tsoukalas et al., 2024), IMO-LeanProofBench (Luong et al., 2025), and CombiBench (Liu et al., 2025).

where a proof usually rests on an explicit construction or count. We compare against pass@16 sampling from the LLMs and five agentic systems: LEAP, Hilbert, AxProver, Aristotle, and ReAct. Our reproduction of each agentic system uses the same base LLM, retains its search strategy, and runs under a matched budget.

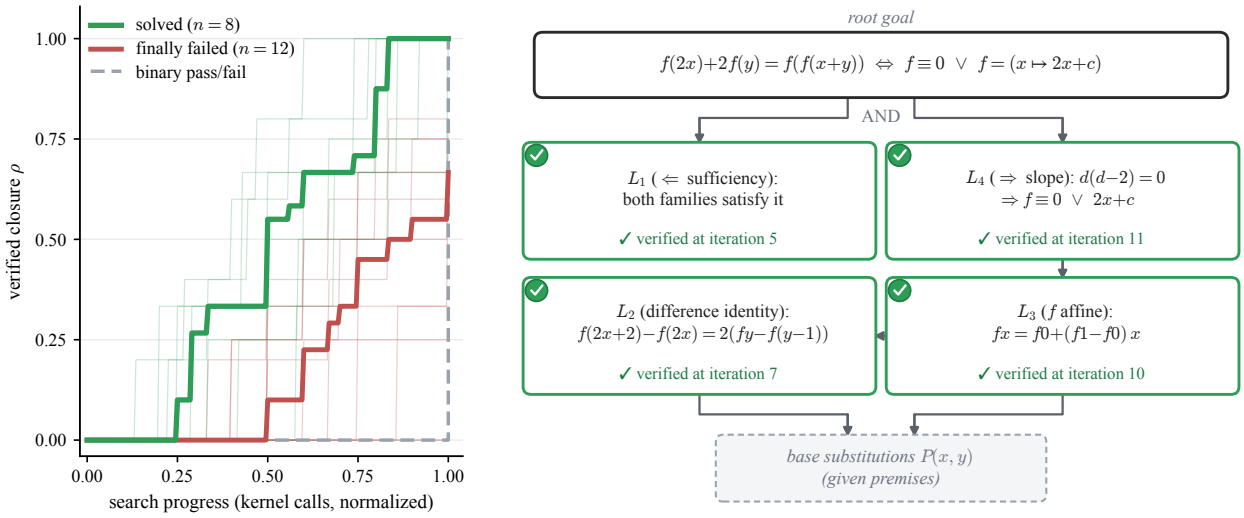
Evaluation metrics. Our primary metric is the solve rate, the fraction of a benchmark whose theorems are proved and pass the Lean 4 kernel verification. A theorem is evaluated as solved only when its final proof matches the benchmark ground truth, contains no unresolved metavariables or placeholders, and passes the Lean kernel. We exclude proofs that rely on `native_decide`, because its code-generation path introduces an axiom outside the standard proof kernel. We apply the same verification to every solution in our evaluation. Every reported solve is independently re-verified against the matched Lean kernel with a restricted `#print axioms` check. Across more than 400 re-verifications, we found 0 false positives.

5.2 Main results

Table 1 reports solve rates on the three benchmarks. Under pass@16 sampling, Claude Opus 4.8 without agentic search solves 0.0% of PutnamBench and IMO-LeanProofBench. The agentic systems use the same base model Claude Opus 4.8, providing the closest matched comparison of their search methods. ProofEvolve has the highest average solve rate at 57.8%, ahead of LEAP (50.5%) and Hilbert (45.9%). It leads PutnamBench at 71.2%, 6.5 points above LEAP, and widens the margin on IMO-LeanProofBench, reaching 53.3% against 36.7% for LEAP. On CombiBench the strongest systems are within one point, LEAP at 50.0% and ProofEvolve at 49.0%. The largest margin appears on IMO-LeanProofBench, whose problems often require proofs assembled from several lemmas. This pattern is consistent with the intended role of graded selection and schema reuse on decomposable problems.

5.3 Dynamics of verified closure ρ

We study how verified closure ρ changes during proof search. ProofEvolve uses ρ in Eq. (9) as a fitness value computed from kernel-accepted edges. Unlike a binary root verdict, it records partial progress once Lean certifies intermediate subgoals. Since Eq. (10) guarantees that solved runs reach $\rho = 1$, we focus on the search trajectory before completion. Figure 2a shows that ρ increases step by step as subgoals are verified, while failed runs plateau below one. Figure 2b shows one solved run in which several lemmas are certified before the root is finally closed. These results show that ρ captures verified intermediate progress that binary feedback cannot represent.



(a) Closure trajectories

(b) One solved proof DAG

Figure 2 Verified closure ρ during search. (a) ρ against evolutionary iterations: solved runs (green) reach 1, failed runs (red) plateau below, and the binary pass/fail signal (dashed) stays at 0 (medians with interquartile bands). **(b)** An accepted proof DAG whose lemma nodes are certified by the kernel at iterations 5, 7, 10, 11, so ρ rises step by step to 1.

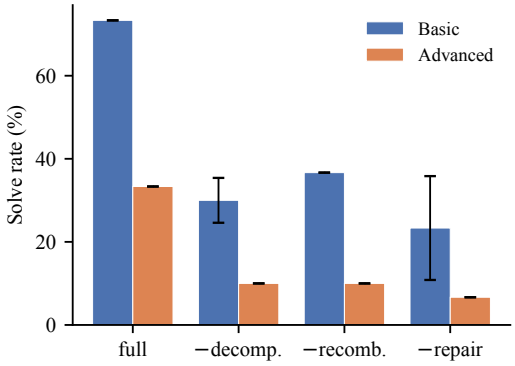


Figure 3 Per-difficulty ablation: every variation operator contributes more on the harder Advanced split than on the Basic split. Error bars show the run-to-run standard deviation across independent reruns.

5.4 Ablation study

We conduct ablation study on how each of the three variation operators affects ProofEvolve. Decomposition breaks a goal into smaller subgoals. Repair fixes a failed step using the error message from Lean 4. Recombination reuses an already proved result to close a new goal. We remove one operator at a time on the 60 IMO-LeanProofBench problems. The base model and the compute budget stay the same, and we repeat each run with five random seeds to ensure statistical stability. As shown in Figure 3, the full system on average solves 32 of the 60 problems, with 22 of 30 on the Basic split and 10 of 30 on the Advanced split. Without decomposition it on average solves 11 (9 Basic, 2 Advanced), without recombination 14 (11 Basic, 3 Advanced), and without repair 9 (7 Basic, 2 Advanced). The drop is larger on the harder Advanced split than on the Basic split. These results show that all three variation operators contribute to the performance of ProofEvolve, with larger effects on the Advanced split.

5.5 Test-time budget scaling

We next test ProofEvolve with open-weight models as the per-target budget grows. The models are Qwen3.5-397B (Qwen Team, 2026a), Qwen3.6-35B (Qwen Team, 2026b), Kimi-K2.6 (Moonshot AI, 2026), GLM-5.1 (Z.ai,

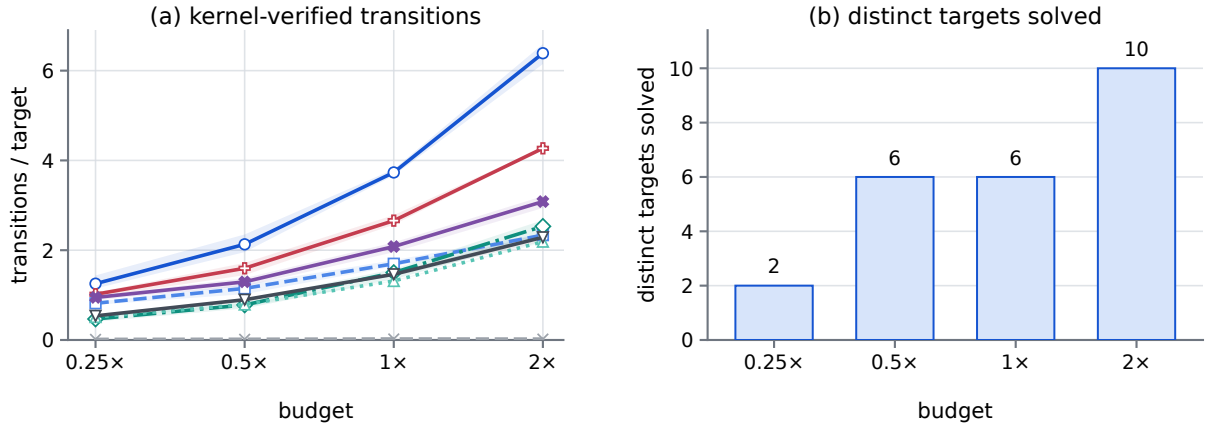
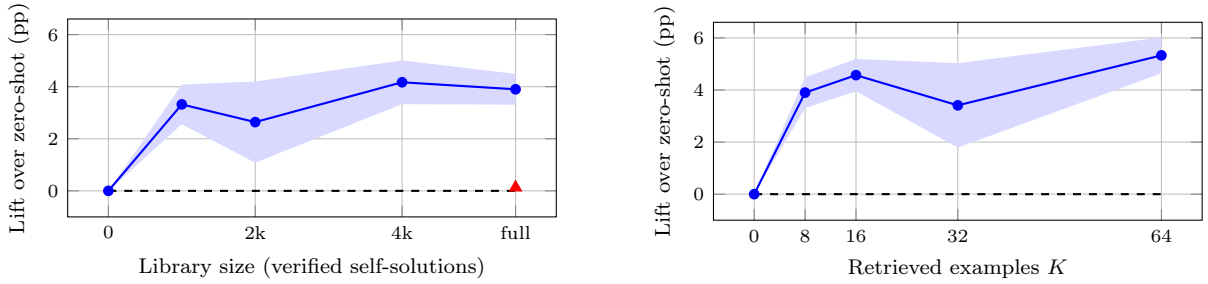


Figure 4 Test-time budget scaling (a) Kernel-verified transitions per target across seeds; (b) Union of distinct targets solved across seeds and configuration. *In (a)*: Qwen3.5 think ○/instant □; Qwen3.6 instant ◊/think △; GLM-5.1 +; Kimi-K2.6 ×; gpt-oss med ▽/low ×.



(a) Library size, at K=8 **(b)** Retrieval depth, full library

Figure 5 Reusing the prover's own verified proofs. Solve-rate lift over zero-shot, in percentage points, on the 744 screened evaluation theorems (three-run mean). ● relevant retrieval; -- zero-shot; ▲ random retrieval from the same library. The band is ±1 run-to-run standard deviation *of the lift*, which is not the standard deviation of the solve rate reported in Table 7. The lift is 0 at an empty library and at K=0, where both conditions reduce to zero-shot.

2026), and gpt-oss-120b (OpenAI, 2025), each run in the inference modes it supports, for eight configurations. A per-target budget caps the model calls, Lean-kernel calls, tokens, and wall-clock time each target may use. Appendix C gives the models, decoding, and the full budget table. Seven of the eight configurations produce more kernel-verified transitions per target as the budget grows (Figure 4a, per-seed values in Appendix D). The one exception, gpt-oss-120b at low reasoning effort, is flat across budgets, confirming that the trend is not merely a by-product of issuing more calls. The final results in Figure 4b shows that the number of distinct targets solved increases monotonically. Appendix D reports the exact per-budget counts. We also provide the solutions found by the models in Appendix E.

5.6 Old Proofs, New Theorems: Verified Reuse Across Problems

A library that grows during evaluation should make later targets easier to prove. We test this at two scales. The controlled study below isolates accumulation in a setting where the dependency structure is known by construction.

Controlled compositional families. To isolate the effect of a library that grows during evaluation, we run a controlled study on synthetic compositional lemma families, where each later target is built from lemmas that earlier targets establish. ProofEvolve solves 19.8% of the targets when the library grows across targets and 7.3% when the library is reset before each target. The growing library therefore solves 2.7× as many targets, with every other component held fixed. The study isolates the inheritance mechanism in this controlled setting,

separate from the benchmark evaluation. Recombination stays sound throughout, because a typed schema discharges a subgoal only when Lean accepts its instantiation, so a mismatched retrieval fails without changing the trusted proof state.

Unseen Lean Workbook theorems. The compositional families are synthetic. We next evaluate cross-problem reuse on real competition-style theorems, using a library built entirely from the prover's own work. We start from Lean Workbook (Ying et al., 2024) statements with the machine-generated proofs released by Goedel-Prover (Lin et al., 2025a). Re-verification under the same kernel and axiom checks retains 20,554 statement and proof pairs, and deduplication leaves 10,968 distinct theorems. We hold out 1,000 for evaluation and use the remaining 9,968 as a source stream. The base model, Qwen3.5-397B-A17B-FP8 (Qwen Team, 2026a), attempts every buildable source theorem once; its 5,546 kernel-accepted proofs form the library. Five independent language-model judges screen the evaluation theorems against their retrieved neighbors, leaving the 744 theorems that fewer than two judges flag. For each evaluation theorem the semantic retriever selects the top-K schemas, and their statements and Lean-accepted proof bodies enter the proposal context. *Zero-shot* omits that context, and *random retrieval* supplies K schemas drawn uniformly from the same library. All conditions share one prompt template, one Lean environment and one decoding profile. We report the mean over three runs, with condition contrasts computed as paired differences across matched runs.

At K=8 the library raises the solve rate from 49.5% to 53.4%, a gain of 3.9 points, while random retrieval from the same library reaches 49.6%: the improvement comes from selecting useful verified work, not from adding examples to the prompt. The gain appears at every library size and retrieval depth we measure. A library of only 1,000 proofs already adds 3.3 points, and the gain reaches 5.3 points at K=64. Figure 5 shows both scaling axes and Table 7 lists every condition. Across the three runs at K=8, relevant retrieval closes 351 theorem instances that zero-shot leaves open, and in 322 of them, or 91.7%, the accepted proof does not reproduce any shown proof verbatim. The library supplies reusable proof structure rather than a catalog of answers.

This study isolates one mechanism. Each condition gives a single whole-proof attempt with no repair and no second sample, so retrieved schemas act as in-context exemplars rather than as typed instantiations composed into a realizer (Eqs. (16)–(18)); the DAG archive, decomposition, repair, and verified closure as a selection signal are all switched off. Holding the search fixed is what makes the library's own contribution measurable. Appendix F records the full setup and Appendix G shows two retrieval-to-proof traces.

6 Conclusion

We introduced ProofEvolve, a neuro-symbolic evolutionary framework that improves through explicit, formally verified proof structures. It represents partial proofs as AND-OR DAGs, ranks them using kernel-grounded verified closure, preserves structurally diverse candidates, and extracts closed sub-DAGs as reusable theorem schemas. The symbolic Lean 4 kernel faithfully checks every proposed variation before it can evolve the internal structured knowledge. Empirically, ProofEvolve achieves the highest average solve rate among the state-of-the-art baselines on three challenging competition-level Lean benchmarks, and on Lean Workbook theorems disjoint from its library a library of the prover's own verified proofs adds about four points over zero-shot, with random retrieval from the same library adding nothing. More broadly, this work paves a concrete step toward recursively self-improving agents that accumulate formal knowledge over time. This insight could shed light on scientific discovery in other scientific domains, such as theoretical physics and chemistry. We hope this work can inspire future research on the field of continual learning for AI-driven scientific discovery.

References

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025. https://arxiv.org/abs/2510.01346.

Anthropic. Claude Haiku 4.5 System Card. https://www.anthropic.com/claude-haiku-4-5-system-card, October 2025. Model ID: `claude-haiku-4-5-20251001`.

Anthropic. Claude Opus 4.8 System Card. https://www.anthropic.com/claude-opus-4-8-system-card, May 2026a. Model ID: `claude-opus-4-8`.

Anthropic. Claude Sonnet 4.6 System Card. https://www.anthropic.com/claude-sonnet-4-6-system-card, February 2026b. Model ID: `claude-sonnet-4-6`.

Ian Berlot-Attwell, Frank Rudzicz, and Xujie Si. Llm library learning fails: A lego-prover case study, 2025. https://arxiv.org/abs/2504.03048.

Roberto Bonola. *Non-Euclidean Geometry: A Critical and Historical Study of Its Development*. Dover Publications, New York, 1955.

Benjamin Breen, Marco Del Tredici, Jacob McCarran, Javier Aspuru Mijares, Weichen Winston Yin, Kfir Sulimany, Jacob M. Taylor, Frank H. L. Koppens, and Dirk Englund. Ax-prover: A deep reasoning agentic framework for theorem proving in mathematics and quantum physics, 2025. https://arxiv.org/abs/2510.12787.

ByteDance Seed. Seed-prover: Deep and broad reasoning for automated theorem proving, 2025. https://arxiv.org/abs/2507.23726.

Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2): 345–363, 1936. doi: 10.2307/2371045.

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. Advancing mathematics by guiding human intuition with ai. *Nature*, 600:70–74, 2021. https://www.nature.com/articles/s41586-021-04086-x.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction (CADE)*, 2021.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning, 2021. https://arxiv.org/abs/2006.08381. PLDI 2021.

Jian Fang, Yican Sun, and Yingfei Xiong. Proof strategy extraction from LLMs for enhancing symbolic provers, 2025. https://arxiv.org/abs/2510.10131.

Google DeepMind. Gemini 3.1 Pro Model Card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, February 2026. API model ID: `gemini-3.1-pro-preview`.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models, 2021. https://arxiv.org/abs/2102.06203. ICLR 2022.

David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik*. Julius Springer, Berlin, 1928.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651, 2026. doi: 10.1038/s41586-025-09833-y. https://www.nature.com/articles/s41586-025-09833-y.

Xingguang Ji, Yahui Liu, Qi Wang, Jingyuan Zhang, Yang Yue, Rui Shi, Chenxi Sun, Fuzheng Zhang, Guorui Zhou, and Kun Gai. Leanabell-Prover-V2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning, 2025. https://arxiv.org/abs/2507.08649.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023. https://arxiv.org/abs/2210.12283. ICLR 2023.

Henry Kautz. The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, 2022.

Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. LeanAgent: Lifelong learning for formal theorem proving, 2024. https://arxiv.org/abs/2410.06209.

Po-Nien Kung, Linfeng Song, Dawsen Hwang, Jinsung Yoon, Chun-Liang Li, Simone Severini, Mirek Olšák, Edward Lockhart, Quoc V. Le, Burak Gokturk, Thang Luong, Tomas Pfister, and Nanyun Peng. LEAP: Supercharging LLMs for formal mathematics with agentic frameworks, 2026. https://arxiv.org/abs/2606.03303.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurelien Rodriguez, and Timothee Lacroix. Hypertree proof search for neural theorem proving, 2022. https://arxiv.org/abs/2205.11491. NeurIPS 2022.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving, 2025a. https://arxiv.org/abs/2502.07640.

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, 2025b. https://arxiv.org/abs/2508.03613.

Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, Zhengfeng Yang, Jujian Zhang, Lihong Zhi, Jia Li, and Zhengying Liu. Combibench: Benchmarking llm capability for combinatorial mathematics, 2025. https://arxiv.org/abs/2505.03171.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. https://aclanthology.org/2025.emnlp-main.1794/.

Moonshot AI. Kimi-K2.6. Hugging Face model card, 2026. https://huggingface.co/moonshotai/Kimi-K2.6. Revision 2755962d.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015. https://arxiv.org/abs/1504.04909.

Yutaka Nagashima. Towards evolutionary theorem proving for isabelle/hol, 2019. https://arxiv.org/abs/1904.08468.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. https://arxiv.org/abs/2506.13131.

OpenAI. gpt-oss-120b. Hugging Face model card, 2025. https://huggingface.co/openai/gpt-oss-120b. Revision b5c939de.

OpenAI. GPT-5.5 System Card. https://openai.com/index/gpt-5-5-system-card/, April 2026. Model ID: `gpt-5.5`.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation, 2024. https://arxiv.org/abs/2407.00695. NeurIPS 2024.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020. https://arxiv.org/abs/2009.03393.

Qwen Team. Qwen3.5-397B-A17B-FP8. Hugging Face model card, 2026a. https://huggingface.co/Qwen/Qwen3.5-397B-A17B-FP8. Revision 9f1f3de9.

Qwen Team. Qwen3.6-35B-A3B. Hugging Face model card, 2026b. https://huggingface.co/Qwen/Qwen3.6-35B-A3B. Revision 53c43178.

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, 2025. https://arxiv.org/abs/2504.21801.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. https://www.nature.com/articles/s41586-023-06924-6.

Ziju Shen, Naohao Huang, Fanyi Yang, Yutong Wang, Guoxiong Gao, Tianyi Xu, Jiedong Jiang, Wanyi He, Pu Yang, Mengzhou Sun, Haocheng Ju, Peihao Wu, Bryan Dai, and Bin Dong. REAL-Prover: Retrieval augmented Lean prover for mathematical reasoning, 2025. https://arxiv.org/abs/2505.20613.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving, 2024. https://arxiv.org/abs/2310.04353. COLM 2024.

The mathlib Community. The Lean mathematical library. In *ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2020.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. https://www.nature.com/articles/s41586-023-06747-5.

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, 2024. https://arxiv.org/abs/2407.11214. NeurIPS 2024 Datasets and Benchmarks.

George Tsoukalas, Anton Kovsharov, Sergey Shirobokov, Anja Surina, Moritz Firsching, Gergely Bérczi, Francisco J. R. Ruiz, Arun Suggala, Adam Zsolt Wagner, Eric Wieser, Lei Yu, Aja Huang, Miklós Z. Horváth, Andrew Ferraiuolo, Henryk Michalewski, Edward Lockhart, Codrut Grosu, Thomas Hubert, Matej Balog, Pushmeet Kohli, and Swarat Chaudhuri. Advancing mathematics research with ai-driven formal proof search, 2026. https://arxiv.org/abs/2605.22763.

Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937. doi: 10.1112/plms/s2-42.1.230.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, and Rose Yu. Hilbert: Recursively building formal proofs with informal reasoning, 2025. https://arxiv.org/abs/2509.22819.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Heng Liao, and Xiaodan Liang. Lego-prover: Neural theorem proving with growing libraries, 2023. https://arxiv.org/abs/2310.00656. ICLR 2024.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. https://arxiv.org/abs/2504.11354.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving, 2025a. https://arxiv.org/abs/2502.03438.

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, 2025b. https://arxiv.org/abs/2509.06493.

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, 2023. https://arxiv.org/abs/2306.15626. NeurIPS 2023 Datasets and Benchmarks.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. https://arxiv.org/abs/2210.03629.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems, 2024. https://arxiv.org/abs/2406.03847.

Z.ai. GLM-5.1-FP8. Hugging Face model card, 2026. https://huggingface.co/zai-org/GLM-5.1-FP8. Revision f396cf80.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents, 2025. https://arxiv.org/abs/2505.22954.

Youyuan Zhang, Jialiang Sun, Hangrui Bi, Chuqin Geng, Wenjie Ma, Zhaoyu Li, and Xujie Si. DreamProver: Evolving transferable lemma libraries via a wake-sleep theorem-proving agent, 2026. https://arxiv.org/abs/2604.26311.

Appendix

A Algorithm

The full evolutionary iteration summarized in Section 4.2 is given below.

Algorithm 1 One evolutionary iteration in ProofEvolve

Input: target queue $\mathcal{Q}$; policy π; kernel $\mathcal{K}$; archives $\{\mathcal{M}_T\}$; schema library $\mathcal{L}$; error store $\mathcal{H}$

Select $T \in \mathcal{Q}$. If $\mathcal{M}_T$ is empty, insert the singleton root DAG.
Sample parent D from $\mathcal{M}_T$ using Eq. (12).
Select $s \in \text{frontier}(D)$ maximizing $\Delta_D(s)$.
Retrieve Mathlib premises $\mathcal{R}_M(s)$ and schemas $\mathcal{R}_{\mathcal{L}}(s)$; form $\mathcal{C}_{\mathcal{L}}(s)$.
if $\mathcal{H}(T, D, s)$ contains a rejected edit and error ϵ **then**
    Sample $\delta \sim \pi(\text{REPAIR} \mid s, D, \mathcal{R}_M(s), \mathcal{C}_{\mathcal{L}}(s), \epsilon)$.
else
    Choose an applicable $o \in \{\text{DECOMPOSE}, \text{RECOMBINE}\}$.
    Sample $\delta \sim \pi(o \mid s, D, \mathcal{R}_M(s), \mathcal{C}_{\mathcal{L}}(s))$.
end if
Compute $D' \leftarrow \text{step}_{\mathcal{K}}(D, \delta)$.
if $D' = \bot$ **then**
    Store the rejected edit and Lean error in $\mathcal{H}(T, D, s)$; **return** with Σ unchanged.
end if
$\mathcal{L} \leftarrow \mathcal{L} \cup \text{Extract}_{\mathcal{K}}(D, D')$.
Update $\mathcal{M}_T[b(D')]$ using Eq. (11).
if $\rho(D') = 1$ and $\mathcal{E}; \Gamma_0 \vdash_{\mathcal{K}} \text{Asm}_{D'}(r) : T$ **then**
    return $\text{Asm}_{D'}(r)$.
end if

B Proofs of Theoretical Results

This appendix states the execution assumptions and derives the invariant and returned-proof results used in Section 4.4.

B.1 Assumptions

Theorem 1 uses the following assumptions.

(A1) The environment $\mathcal{E}$ and kernel $\mathcal{K}$ are fixed throughout the execution. Every target that occurs in the execution is well formed in $\mathcal{E}$. Every initial DAG is finite, acyclic, rooted at $(\Gamma_0 \vdash T)$, and each accepted edge satisfies Eq. (2).

(A2) Every initial library entry $(\ell, \pi_\ell) \in \mathcal{L}_0$ satisfies $\mathcal{E} \vdash_{\mathcal{K}} \pi_\ell : \ell$.

(A3) Every later DAG is produced by $\text{step}_{\mathcal{K}}$ in Eq. (7). Every later library entry is produced by $\text{Extract}_{\mathcal{K}}$, which returns a pair (ℓ, π_ℓ) only if Lean elaborates the resulting declaration without unresolved metavariables and verifies $\mathcal{E} \vdash_{\mathcal{K}} \pi_\ell : \ell$. If abstraction, elaboration, or kernel verification fails, no schema is returned. Archive and library updates follow Algorithm 1.

B.2 Kernel-grounded invariance

Let $\mathfrak{D}_0$ contain the initial DAGs; for $t > 0$, $\mathfrak{D}_t$ also contains every challenger accepted during transitions $0, \ldots, t-1$, including challengers later discarded by archive comparison.

Theorem 1 (Invariance of kernel-grounded state)**.** *Under the formal assumptions in Appendix B, every finite execution $\mathcal{S}_0 \to \cdots \to \mathcal{S}_n$ of Algorithm 1 satisfies, for each $0 \le t \le n$:*

(I1) *every accepted edge in every $D \in \mathfrak{D}_t$ has a checked realizer of the form in Eq. (2);*
(I2) *every closed node s in every $D \in \mathfrak{D}_t$ satisfies $\mathrm{Asm}_D(s) \in \mathsf{Prf}_{\mathcal{E}}(s)$; and*
(I3) *every $(\ell, \pi_\ell) \in \mathcal{L}_t$ satisfies $\mathcal{E} \vdash_{\mathcal{K}} \pi_\ell : \ell$.*

For every $0 \le t < n$, $\mathcal{L}_t \subseteq \mathcal{L}_{t+1}$. If transition t rejects its proposal, then $\Sigma_{t+1} = \Sigma_t$.

Proof. We use induction on the transition index t. Assumption (A1) establishes (I1) at $t = 0$, and (A2) establishes (I3).

To derive (I2), fix $D \in \mathfrak{D}_0$ and a closed node s. Write $\mathrm{win}_D(s) = (s; s_1, \ldots, s_k)$ and define

$$h_D(s) = \begin{cases} 0, & k = 0, \\ 1 + \max\limits_{1 \le i \le k} h_D(s_i), & k > 0. \end{cases} \tag{19}$$

Acyclicity makes $h_D(s)$ finite. If $h_D(s) = 0$, then (I1) gives

$$\mathrm{Asm}_D(s) = F_{\mathrm{win}_D(s)}(\star) \in \mathsf{Prf}_{\mathcal{E}}(s). \tag{20}$$

For $h_D(s) > 0$, every s_i has smaller height. The inner induction gives $p_i = \mathrm{Asm}_D(s_i) \in \mathsf{Prf}_{\mathcal{E}}(s_i)$, so

$$\mathrm{Asm}_D(s) = F_{\mathrm{win}_D(s)}(p_1, \ldots, p_k) \in \mathsf{Prf}_{\mathcal{E}}(s). \tag{21}$$

This proves (I2) at $t = 0$.

Assume (I1)–(I3) at index t. If $\mathrm{step}_{\mathcal{K}}(D, \delta) = \bot$, Algorithm 1 updates only the error store. Therefore

$$\Sigma_{t+1} = \Sigma_t, \qquad \mathcal{L}_{t+1} = \mathcal{L}_t, \qquad \mathfrak{D}_{t+1} = \mathfrak{D}_t. \tag{22}$$

All three invariants follow immediately.

Suppose instead that $\mathrm{step}_{\mathcal{K}}(D, \delta) = D'$. By Eq. (7), $D \preceq D'$ and the new edge has a checked realizer. All old edges retain their realizers, so (I1) holds for

$$\mathfrak{D}_{t+1} = \mathfrak{D}_t \cup \{D'\}. \tag{23}$$

Apply the height induction in Eqs. (19)–(21) to every closed node of D'. Its witnessing edges are either old edges, covered by the outer induction hypothesis, or the new edge, covered by Eq. (7). Hence (I2) holds for D' and remains true for every DAG in $\mathfrak{D}_t$.

The library update is

$$\mathcal{L}_{t+1} = \mathcal{L}_t \cup \mathrm{Extract}_{\mathcal{K}}(D, D') \supseteq \mathcal{L}_t. \tag{24}$$

Every extracted pair satisfies Eq. (14), so (I3) is preserved. Equation (11) stores either D' or the previous valid incumbent. This completes the outer induction. □

B.3 Validity of returned proofs

Corollary 1 (Validity of returned proofs)**.** *Under the assumptions of Theorem 1, if ProofEvolve returns p from $D' \in \mathfrak{D}_n$ for a target T with root $r = (\Gamma_0 \vdash T)$, then $\mathcal{E}; \Gamma_0 \vdash_{\mathcal{K}} p : T$.*

Proof. The return guard in Algorithm 1, the equivalence in Eq. (10), and Theorem 1(I2) give

$$\rho(D') = 1 \Longrightarrow \mathrm{Closed}_{D'}(r) \Longrightarrow \mathrm{Asm}_{D'}(r) \in \mathsf{Prf}_{\mathcal{E}}(r). \tag{25}$$

The algorithm returns $p = \mathrm{Asm}_{D'}(r)$. By Eq. (1),

$$p \in \mathsf{Prf}_{\mathcal{E}}(\Gamma_0 \vdash T) \Longrightarrow \mathcal{E}; \Gamma_0 \vdash_{\mathcal{K}} p : T. \tag{26}$$

□

Model	Developer	Engine	TP	Mode	temp.	top-p	top-k
Kimi-K2.6	Moonshot AI	SGLang	8	instant	0.6	0.95	–
Qwen3.5-397B-A17B-FP8	Alibaba	vLLM	4	instant	0.7	0.80	20
				thinking	0.6	0.95	20
Qwen3.6-35B-A3B	Alibaba	vLLM	1	instant	0.7	0.80	20
				thinking	1.0	0.95	20
GLM-5.1-FP8	Z.ai	SGLang	8	instant	1.0	–	–
gpt-oss-120b	OpenAI	SGLang	1	low/medium effort	1.0	1.0	0

Table 2 Self-hosted fixed-weight models and their serving and decoding settings. TP denotes tensor parallelism per replica, and dashes mark parameters left at their vendor defaults. Qwen instant modes use a presence penalty of 1.5, compared with 0 in thinking mode. For gpt-oss-120b, we vary the reasoning effort.

Budget	Model calls	Lean calls	Tokens	Wall (s)
0.25×	3	15	100K	450
0.5×	6	30	200K	900
1×	12	60	400K	1,800
2×	24	120	800K	3,600

Table 3 Per-target composite budget profiles. Each row gives four hard caps. "Tokens" is the combined input+output allowance; the per-call output cap remains 32,768.

C Setup of Open-weight Models

This appendix gives the setup for the test-time compute scaling study in Section 5.5. The study uses the ProofEvolve mechanism from Section 4 and varies the base model and composite per-target budget. Each reported proof elaborates in the frozen Lean 4 environment with the matched Mathlib commit. It contains no unresolved metavariables or placeholders, does not use `native_decide`, and passes an independent `#print axioms` check. We use seeds $\{19, 36, 65\}$.

Target set. The scheduled evaluation manifest contains 485 targets from the benchmarks in Table 1: PutnamBench (326), IMO-LeanProofBench (60), and CombiBench (99). Each target is scheduled with all three seeds.

Models and serving. All open-weight serving ran on the computation nodes with 8× NVIDIA B200 GPUs (192 GB HBM each), dual-socket Intel Xeon hosts (224 vCPUs, about 3.9 TB RAM), running Ubuntu 22.04.5 LTS (kernel `6.8.0-1040-gcp`) with CUDA 12.8. We serve under Python 3.11.15 with two engines: vLLM 0.19.0 (PyTorch 2.10.0+cu128) for the Qwen models and SGLang 0.5.10 (PyTorch 2.9.1+cu128) for GLM-5.1, Kimi-K2.6, and gpt-oss-120b. Per-model tensor parallelism is listed in Table 2. Proof verification uses Lean 4.29.1 with Mathlib commit `5e932f97` and pantograph 0.3.15. Proprietary baselines (Claude Opus 4.8) are accessed through the vendor API. The headline results in Table 1 therefore use API inference plus local Lean kernel verification and do not consume the B200 cluster.

Decoding. Table 2 gives the decoding parameters, which remain fixed across budget scales. The per-call output cap is 32,768 tokens for every run.

Composite budget. The profiles $\{0.25, 0.5, 1, 2\}\times$ jointly scale four hard caps relative to the 1× reference in Table 3: model calls, Lean calls, tokens, and wall-clock time. The mechanism, decoding parameters, and verification procedure remain fixed. Because all four caps change together, the study does not isolate the effect of any one resource.

Model	Mode	0.25×	0.5×	1×	2×
Qwen3.5-397B	thinking	1.44/1.15/1.18	2.35/1.96/2.08	3.80/3.66/3.73	6.41/6.15/6.59
Qwen3.5-397B	instant	0.82/0.78/0.85	1.19/1.02/1.23	1.64/1.78/1.67	2.26/2.37/2.37
Qwen3.6-35B	instant	0.48/0.51/0.41	0.70/0.89/0.75	1.35/1.58/1.57	2.27/2.67/2.65
Qwen3.6-35B	thinking	0.50/0.51/0.53	0.81/0.82/0.75	1.31/1.32/1.31	2.21/2.19/2.17
GLM-5.1	instant	1.05/0.95/1.06	1.72/1.44/1.63	2.57/2.63/2.78	4.26/4.27/4.29
Kimi-K2.6	instant	0.97/0.89/0.98	1.38/1.20/1.31	1.95/2.12/2.17	3.20/2.94/3.11
gpt-oss-120b	med.	0.56/0.49/0.57	0.90/0.90/0.89	1.42/1.47/1.50	2.19/2.28/2.39
gpt-oss-120b	low	0.01/0.02/0.01	0.01/0.01/0.02	0.03/0.01/0.02	0.02/0.02/0.02

Table 4 Mean kernel-verified transitions per target, with entries ordered as **seed 19 / seed 36 / seed 65**. Figure 4a plots the target-weighted mean across the three seeds. These counts measure cumulative verified search activity, not unique proofs or solve rates.

Model	Mode	0.25×	0.5×	1×	2×
Qwen3.5-397B	thinking	3	4	5	9
Qwen3.5-397B	instant	3	3	3	6
Qwen3.6-35B	instant	3	3	4	3
Qwen3.6-35B	thinking	4	3	3	4
GLM-5.1	instant	3	4	5	3
Kimi-K2.6	instant	3	7	9	11
gpt-oss-120b	medium	6	5	6	6
gpt-oss-120b	low	3	3	0	2
All eight configurations: solve events		28	32	35	44
Union of solved targets		2	6	6	10

Table 5 Final kernel-verified outcomes across seeds $\{19, 36, 65\}$. A solve event is one run for a particular model, mode, seed, and target that returns a valid proof. Repeated solutions of the same target count separately. The last row counts target identifiers solved at least once. Budget columns correspond to separate runs rather than cumulative prefixes.

D Results of Test-Time Budget Scaling

Search activity. Table 4 reports mean kernel-verified transitions per target for each seed in $\{19, 36, 65\}$. Figure 4a plots the target-weighted mean across the three seeds. Among recorded outcomes, seven of the eight configurations increase monotonically with budget in every seed. Low-effort gpt-oss-120b remains near zero. The number of recorded outcomes decreases for some runs in thinking mode at larger budgets. For example, Qwen3.5-397B in thinking mode has $n = 1441/1428/1359/1271$ recorded target–seed outcomes at $0.25/0.5/1/2\times$. The reported values are target-weighted over recorded outcomes.

Parser coverage. The proof-state S-expression parser deliberately does not support the AST forms `:mv`, `:mvd`, `:subst`, and `:proj`. When the parser encounters one of these forms, the runner exits before writing a terminal outcome. These exits occur more often in deeper searches, so missing outcome records become more frequent as the budget grows. Means computed only from recorded outcomes may therefore be biased upward at larger budgets. We treat the curves as descriptive search activity, not as unbiased estimates for the full manifest or measures of proposal efficiency.

Solves. Table 5 reports both solve events and distinct target coverage. Between $0.25\times$ and $2\times$, the number of solve events rises from 28 to 44, while the union of solved targets rises from 2 to 10. Kimi-K2.6 and the two Qwen3.5 modes account for most of the endpoint increase. Four targets are solved at $2\times$ but not at a lower budget: `brualdi_ch10_31`, `brualdi_ch1_10`, `hackmath_4`, and `putnam_1977_a5`. The budget arms are separate stochastic runs, and the number of recorded outcomes varies with budget. These counts describe observed coverage; they do not establish monotone per-model scaling or isolate the effect of any one resource.

Target	Bench	Model (mode, budget)	Trans.	Calls	Lean	Representative mechanism
`putnam_2012_a2`	Putnam	Qwen3.5-397B (thinking, 2×)	7	17	40	decomposition and derived identity lemma
`putnam_1977_a5`	Putnam	Qwen3.5-397B (thinking, 2×)	7	23	59	ProofLib schema reuse
`putnam_2001_a1`	Putnam	Qwen3.5-397B (thinking, 2×)	3	5	21	hypothesis instantiation and rewrite
`putnam_1988_b1`	Putnam	Qwen3.5-397B (thinking, 2×)	2	5	10	explicit construction and `ring`
`hackmath_4`	Combi	Kimi-K2.6 (instant, 2×)	1	3	8	`IsLeast` decomposition and pigeonhole
`brualdi_ch1_10`	Combi	Kimi-K2.6 (instant, 2×)	4	14	33	order-2 impossibility via `omega`
`brualdi_ch7_7`	Combi	Kimi-K2.6 (instant, 2×)	3	3	28	`Int.gcd_fib` rewrite
`brualdi_ch8_6`	Combi	Kimi-K2.6 (instant, 2×)	1	3	11	induction on the summation
`brualdi_ch14_33`	Combi	Kimi-K2.6 (instant, 2×)	1	1	10	single-lemma rewrite (`cycleType_inv`)
`brualdi_ch2_11`	Combi	GLM-5.1 (instant, 1×)	1	2	7	kernel `decide`
`brualdi_ch10_31`	Combi	Qwen3.6-35B (thinking, 2×)	2	22	30	witness and kernel `decide`

Table 6 The 11 distinct targets solved in the open-weight runs. For one representative run per target, "Trans." gives kernel-verified transitions, "Calls" gives model calls, and "Lean" gives kernel calls. Every listed run reaches $\rho = 1$. Medium-effort gpt-oss-120b also solves `brualdi_ch14_33` and `brualdi_ch7_7`. Appendix E gives the corresponding proofs.

Solved-target inventory. Across all budgets and seeds, the open-weight runs solve 11 distinct targets: 4 from PutnamBench and 7 from CombiBench. None is from IMO-LeanProofBench. Table 6 gives the model, budget, and search statistics for one representative run per target. All five models solve `brualdi_ch14_33`. The listed solves for `putnam_2012_a2` and `putnam_1977_a5` occur only at the larger budgets.

Re-verification. We re-verify every solve in the frozen Lean 4.29.1 environment with Mathlib commit `5e932f97`. Each returned proof term is elaborated from scratch, and `#print axioms` enumerates its axiom dependencies. All 139 kernel-verified solve events have a stored closing proof, with no reported $\rho = 1$ lacking one. Every proof depends only on {`propext`, `Classical.choice`, `Quot.sound`}, and none uses `native_decide`. Re-verification found 0 false positives.

Verified substrate. In addition to Mathlib, the ProofEvolve environment $\mathcal{E}$ contains the library of 158 kernel-verified lemmas. The library has no occurrences of `sorry` and introduces no axioms beyond the standard three. A proof that uses one of these lemmas therefore has the same axiom footprint as a proof built directly on Mathlib. During re-verification, we inline the library so that `#print axioms` checks its dependencies transitively. The proof of `putnam_1977_a5` in Appendix E uses one lemma from this library.

Comparison with agentic baselines. We also evaluate pass@16, escalating whole-proof search, ReAct, LEAP, Hilbert, and ProofEvolve on a fixed random subset shared across models. All methods solve few targets with these open-weight models, so we draw no quantitative cross-method conclusion from this subset. Verified closure and kernel-verified transition counts are specific to ProofEvolve's DAG and do not support a comparison with the other methods. Table 1, which uses Claude Opus 4.8 on the full benchmarks, provides the matched solve-rate comparison.

E Kernel-Certified Open-Weight Proofs on PutnamBench and CombiBench

This appendix gives the kernel-certified proofs for all 11 distinct open-weight solves in Table 6. We verify them as described in Appendix D. Each theorem name is its benchmark target identifier. The caption records the solving model and the kernel-verified transitions and model calls for one solving run.

E.1 PutnamBench

Listing 1: putnam_2012_a2: Qwen3.5-397B (thinking, 2×), 7 transitions / 17 calls

```
import Mathlib
import ProofLib

open Matrix

theorem putnam_2012_a2
(S : Type*) [CommSemigroup S]
(a b c : S)
(hS : ∀ x y : S, ∃ z : S, x * z = y)
(habc : a * c = b * c)
: a = b := by
  obtain ⟨z, hz⟩ := hS c a
  obtain ⟨e, he⟩ := hS c c
  have h_id : ∀ x : S, x * e = x := by
    intro x
    obtain ⟨w, hw⟩ := hS c x
    calc
      x * e = (c * w) * e := by rw [hw]
      _ = c * (w * e) := by rw [mul_assoc]
      _ = c * (e * w) := by rw [mul_comm w e]
      _ = (c * e) * w := by rw [mul_assoc]
      _ = c * w := by rw [he]
      _ = x := by rw [hw]
  obtain ⟨d, hd⟩ := hS c e
  exact calc
    a = a * e := by rw [h_id a]
    _ = a * (c * d) := by rw [hd]
    _ = (a * c) * d := by rw [mul_assoc]
    _ = (b * c) * d := by rw [habc]
    _ = b * (c * d) := by rw [mul_assoc]
    _ = b * e := by rw [hd]
    _ = b := by rw [h_id b]
```

right cancellation in a commutative semigroup satisfying the stated divisibility condition. The proof first establishes $\forall x,\ x * e = x$ and then uses this identity in the cancellation argument.

Listing 2: putnam_1977_a5: Qwen3.5-397B (thinking, 2×), 7 transitions / 23 calls

```
import Mathlib
import ProofLib

open RingHom Set Nat

theorem putnam_1977_a5
(p m n : ℕ)
(hp : Nat.Prime p)
(hmgen : m ≥ n)
: (choose (p * m) (p * n) ≡ choose m n [MOD p]) := by
  rw [Nat.ModEq]
  haveI : Fact (Nat.Prime p) := ⟨hp⟩
  rw [Choose.choose_modEq_choose_mod_mul_choose_div_nat]
  simp [Nat.mul_mod, Nat.mul_div_cancel_left, Nat.choose_zero_right, Nat.mod_eq_of_lt]
  simp [Nat.mul_div_cancel_left, Nat.Prime.pos hp]
```

the Lucas-type congruence $\binom{pm}{pn} \equiv \binom{m}{n} \pmod{p}$. The proof applies a p-adic binomial congruence lemma from ProofLib and finishes with simp.

Listing 3: putnam_1988_b1: Qwen3.5-397B (thinking, 2×), 2 transitions / 5 calls

```
import Mathlib
import ProofLib

open Set Filter Topology

theorem putnam_1988_b1
: ∀ a ≥ 2, ∀ b ≥ 2, ∃ x y z : ℤ, x > 0 ∧ y > 0 ∧ z > 0 ∧ a * b = x * y + x * z + y * z + 1 := by
  intro a ha b hb; use 1, a - 1, b - 1; constructor; norm_num; constructor; linarith; constructor; linarith; ring
```

every product ab with $a, b \geq 2$ equals $xy + xz + yz + 1$ for positive integers x, y, z. The proof sets $x = 1$, $y = a - 1$, and $z = b - 1$, then finishes with `linarith` and `ring`.

Listing 4: putnam_2001_a1: Qwen3.5-397B (thinking, 2×), 3 transitions / 5 calls

```
import Mathlib
import ProofLib

open Topology Filter

theorem putnam_2001_a1
(S : Type*)
[Mul S]
(hS : ∀ a b : S, (a * b) * a = b)
: ∀ a b : S, a * (b * a) = b := by
  intro a b
  have h := hS b a; have h2 := hS (b * a) b; rw [h] at h2; exact h2
```

if a magma satisfies $(a * b) * a = b$ for all a, b, then $a * (b * a) = b$. The proof instantiates the hypothesis twice and rewrites once.

E.2 CombiBench

Listing 5: hackmath_4: Kimi-K2.6 (instant, 2×), 1 transitions / 3 calls

```
import Mathlib

theorem hackmath_4 : IsLeast {n | ∀ f : Fin n → Fin 12, ∃ a b, a ≠ b ∧ f a = f b} ((13) : ℕ ) := by
  constructor
  · -- Show 13 is in the set
    intro f
    have h1 : Fintype.card (Fin 13) = 13 := by simp
    have h2 : Fintype.card (Fin 12) = 12 := by simp
    have h3 : Fintype.card (Fin 13) > Fintype.card (Fin 12) := by rw [h1, h2]; norm_num
    obtain ⟨a, b, hab, heq⟩ := Fintype.exists_ne_map_eq_of_card_lt f h3
    exact ⟨a, b, hab, heq⟩
  · -- Show 13 is the least element in the set
    intro n hn
    by_contra h
    push_neg at h
    have h4 : n ≤ 12 := by omega
    have h5 : ∃ f : Fin n ↪ Fin 12, True := by
      have h6 : Fintype.card (Fin n) ≤ Fintype.card (Fin 12) := by
        simp
        omega
      have h7 : Nonempty (Fin n ↪ Fin 12) := Function.Embedding.nonempty_of_card_le h6
      obtain ⟨f⟩ := h7
      exact ⟨f, trivial⟩
    obtain ⟨f, _⟩ := h5
    have h7 : ∀ a b, a ≠ b → f a ≠ f b := by
      intro a b hab
      exact f.injective.ne hab
```

Listing 5: hackmath_4: Kimi-K2.6 (instant, 2×), 1 transitions / 3 calls (continued)

```
  have h8 := hn f
  obtain ⟨a, b, hab, heq⟩ := h8
  have h9 : f a ≠ f b := h7 a b hab
  contradiction
```

13 is the least group size that forces two people into the same month. The proof establishes membership with `Fintype.exists_ne_map_eq_of_card_lt` and minimality by constructing an embedding for $n \leq 12$.

Listing 6: brualdi_ch1_10: Kimi-K2.6 (instant, 2×), 4 transitions / 14 calls

```
import Mathlib

structure IsMagicSquare {n : ℕ} (M : Matrix (Fin n) (Fin n) ℕ) : Prop where
  mem : ∀ i j, M i j ∈ Finset.Icc 1 (n * n)
  pairwise : ∀ i j i' j', i ≠ i' ∨ j ≠ j' → M i j ≠ M i' j'
  same_sum : ∃ s, (∀ i, ∑ j, M i j = s) ∧ (∀ j, ∑ i, M i j = s) ∧ (∑ i, M i i.rev = s) ∧ ∑ i, M i i = s

theorem brualdi_ch1_10 : ¬∃ (M : Matrix (Fin 2) (Fin 2) ℕ), IsMagicSquare M := by
  intro h
  rcases h with ⟨M, hM⟩
  rcases hM with ⟨h_rows, h_cols, h_diag, h_anti, h_pairwise⟩
  have h1 := h_rows 0 0
  have h2 := h_rows 0 1
  have h3 := h_rows 1 0
  have h4 := h_rows 1 1
  have h5 := h_cols 0 0 0 1 (Or.inr (by decide))
  have h6 := h_cols 0 0 1 0 (Or.inl (by decide))
  have h7 := h_cols 0 0 1 1 (Or.inl (by decide))
  have h8 := h_cols 0 1 1 0 (Or.inl (by decide))
  have h9 := h_cols 0 1 1 1 (Or.inl (by decide))
  have h10 := h_cols 1 0 1 1 (Or.inr (by decide))
  simp [Finset.mem_Icc] at h1 h2 h3 h4
  have h_row0 := h_anti 0
  have h_row1 := h_anti 1
  have h_col0 := h_pairwise.1 0
  have h_col1 := h_pairwise.1 1
  have h_diag1 := h_pairwise.2.2
  have h_anti_diag := h_pairwise.2.1
  simp [Fin.sum_univ_two, Fin.rev] at h_row0 h_row1 h_col0 h_col1 h_diag1 h_anti_diag
  omega
```

no order-2 magic square exists. The proof states the range, distinctness, and row, column, and diagonal sum constraints, then solves the resulting integer system with `omega`.

Listing 7: brualdi_ch7_7: Kimi-K2.6 (instant, 2×), 3 transitions / 3 calls

```
import Mathlib

theorem brualdi_ch7_7 (m n d : ℕ+) (hmd : d = Nat.gcd m n) :
    Nat.gcd (Nat.fib m) (Nat.fib n) = Nat.fib d := by
  have h1 : (Nat.fib ↑m).gcd (Nat.fib ↑n) = Nat.fib (Int.gcd (↑m : ℤ) (↑n : ℤ)) := by rw [← Int.gcd_fib (↑m : ℤ) (↑n : ℤ)
      ]; simp
  rw [hmd]
  simp [hmd] at h1 ⊢; exact h1
```

$\gcd(F_m, F_n) = F_{\gcd(m,n)}$ for Fibonacci numbers. The proof converts between the $\mathbb{N}$ and $\mathbb{Z}$ formulations and applies the library identity.

Listing 8: brualdi_ch8_6: Kimi-K2.6 (instant, 2×), 1 transitions / 3 calls

```
import Mathlib

theorem brualdi_ch8_6 (n : ℕ) (h : ℕ → ℝ) (h' : ∀ i, h i = 2 * i ^ 2 - i + 3) :
    ∑ i ∈ Finset.range (n + 1), h i = ((fun n => ((n + 1) * (4 * n ^ 2 - n + 18) / 6)) : ℕ → ℝ ) n := by
  induction n with
  | zero =>
    simp [Finset.sum_range_succ, h']
    all_goals norm_num
  | succ n ih =>
    rw [Finset.sum_range_succ, ih]
    simp [h']
    ring_nf
    <;> field_simp
    <;> ring_nf
    <;> norm_num
    <;> ring
```

the closed form $\sum_{k=0}^{n}(2k^2-k+3)=\frac{(n+1)(4n^2-n+18)}{6}$, proved by induction.

Listing 9: brualdi_ch14_33: Kimi-K2.6 (instant, 2×), 1 transitions / 1 calls

```
import Mathlib

theorem brualdi_ch14_33 {α : Type*} [Fintype α] [DecidableEq α] (σ : Equiv.Perm α) :
    σ.cycleType = σ⁻¹.cycleType := by
  rw [Equiv.Perm.cycleType_inv]
```

a permutation and its inverse have the same cycle type. The proof applies the library identity `Equiv.Perm.cycleType_inv`; all five models solve this target.

Listing 10: brualdi_ch2_11: GLM-5.1 (instant, 1×), 1 transitions / 2 calls

```
import Mathlib

open Finset

theorem brualdi_ch2_11 :
    ((Icc (1 : ℕ) 20).powersetCard 3 |>.filter (fun S => ∀ a ∈ S, a - 1 ∉ S ∧ a + 1 ∉ S)).card =
    ((816) : ℕ ) := by
  set_option maxRecDepth 1000000 in decide
```

there are 816 size-3 subsets of $\{1,\dots,20\}$ with no two consecutive elements. The kernel tactic `decide` verifies the count without `native_decide`.

Listing 11: brualdi_ch10_31: Qwen3.6-35B (thinking, 2×), 2 transitions / 22 calls

```
import Mathlib

def isDifferenceSet (n : ℕ) (B : Finset (ZMod n)) : Prop :=
  ∃ k, ∀ x : (ZMod n),  x ≠ 0 → ∑ i ∈ B, ∑ j ∈ B \ {i}, List.count x [i - j] = k

theorem brualdi_ch10_31 : isDifferenceSet 21 {0, 3, 4, 9, 11} := by
  dsimp only [isDifferenceSet]
  use 1; decide
```

$\{0,3,4,9,11\}$ is a difference set in $\mathbb{Z}_{21}$. The proof supplies the multiplicity witness and uses the kernel tactic `decide`, not `native_decide`.

F Detailed Setup for the Lean Workbook Study

This appendix records the corpus, library construction, screening, retrieval and evaluation used in Section 5.6.

F.1 Corpus and verification

We use theorem statements from Lean Workbook (Ying et al., 2024) and the machine-generated proofs released by Goedel-Prover (Lin et al., 2025a). We re-elaborate every candidate in a fixed Lean 4 environment (de Moura and Ullrich, 2021) with a matched Mathlib commit (The mathlib Community, 2020), under the same acceptance criteria used throughout the paper: an accepted proof has no unresolved metavariables or placeholders, passes the Lean kernel at the stated type, and a restricted `#print axioms` check must show dependencies only on `propext`, `Classical.choice` and `Quot.sound`. We reject proofs that use `native_decide` because its code generation path introduces an axiom outside the standard kernel. Each candidate runs in a fresh process with a 300-second timeout. This verification retains 20,554 statement and proof pairs.

F.2 Deduplication and the evaluation split

Lean Workbook contains many restatements of the same theorem, so we deduplicate before splitting. We generate candidate pairs with MinHash LSH over statement shingles using 128 permutations, 32 bands of 4 rows, seed 20260806 and an approximate threshold of 0.42. This produces 2,362,946 candidate pairs. We confirm each pair with direct similarity tests or a guarded signature match. The direct tests use alpha equivalence of the elaborated statement, statement n-gram Jaccard above 0.8 and docstring n-gram Jaccard above 0.8. The guarded match requires signature Jaccard above 0.95 over at least five symbols and statement Jaccard above 0.6. We disable the dense embedding channel for deduplication because transitive embedding matches can join distinct theorems into one cluster. The exact tests retain 26,330 edges and 5,216 multi-theorem clusters. The largest cluster contains 106 theorems. Removing 6,935 alpha-exact duplicates and 2,651 near duplicates leaves 10,968 representatives, a 46.6% reduction. A fixed seed assigns 1,000 theorems to evaluation and 9,968 to the source stream. A post-split audit checks the boundary again for residual near duplicates.

F.3 Leakage screening

The exact tests can still miss the same theorem written in a different form. Five independent language-model judges therefore screen each evaluation theorem against its 64 nearest library neighbors. A judge flags a residual near duplicate when a neighbor states the same theorem up to renaming or a change of constants. We record every verdict and its reason, then exclude a theorem when at least two judges flag it. The screen removes 256 theorems, or 25.6%, and leaves the 744 used throughout the study. Screening occurs before condition outcomes are compared, and every condition uses the same evaluation set.

The panel is five Claude models spanning four releases (`claude-opus-5`, `claude-opus-4.8`, `claude-opus-4.7`, `claude-opus-4.6` and `claude-sonnet-5`), each queried once per evaluation theorem at medium reasoning effort. Verdicts are independent across models but not across model families. Flag counts over the 1,000 candidates are 624, 120, 57, 37, 56 and 106 for zero through five flags. Agreement is therefore bimodal: 730 theorems, or 73.0%, receive a unanimous verdict, and only 94 fall in the two-to-three-flag band where the threshold is decisive. Raising the threshold to three flags would retain 801 theorems instead of 744. Prompt 1 gives the template; each judge sees only the evaluation theorem and its retrieved neighbors.

Prompt 1: Leakage judge, one call per evaluation theorem

```
You are a LEAKAGE auditor for a held-out Lean theorem-proving benchmark. Read
ONLY the record for this evaluation theorem (a single JSON object with
holdout_statement, holdout_docstring, and a "retrieved" array -- each item has
statement + proof_body + docstring). Do NOT read any other file.

Rule LEAK if ANY retrieved item is a NEAR-DUPLICATE of the holdout (the same
problem up to variable renaming / constant changes / trivial rewrite) OR
directly contains the holdout's answer or a proof that would transfer by
copying (memorization, not reasoning).
```

Prompt 1: Leakage judge, one call per evaluation theorem (continued)

```
Rule CLEAN if the retrieved items are only related-but-distinct (same technique
/ topic, genuinely different problem).

Give the verdict + a one-sentence reason.
```

F.4 Verified theorem schema library

The base model attempts every buildable theorem in the 9,968-theorem source stream once under the same one-shot budget used for evaluation. Target files cannot be built for 384 theorems. Of the remaining 9,584, the kernel and axiom audit accept 5,546 proofs. The self-solve rate is 57.9% over scoreable theorems and 55.6% over the full source stream. These 5,546 theorem and proof pairs form the persistent verified theorem schema library. Every entry contains a proof written by the base model and checked by Lean. No human proof or released Goedel-Prover proof enters the library. Unlike ProofLib, whose lemmas are installed in $\mathcal{E}$ and therefore enter the axiom footprint of any proof that uses them (Appendix D), this library is a retrieval corpus only: its entries never enter the Lean environment and never appear in any proof's axiom footprint. The library-size conditions use nested subsets drawn once with a fixed seed, so the 1,000-proof library is contained in the 2,000-proof library, which is contained in the 4,000-proof library.

F.5 Semantic retrieval

The semantic retriever embeds theorem statements with `bge-large-en-v1.5`. Raw cosine similarities occupy a narrow band because the corpus contains many competition-style algebra and inequality theorems. We subtract the library mean from each vector before computing cosine similarity. The encoder uses asymmetric instructions. Evaluation queries carry the prefix `Represent this sentence for searching relevant passages:`, while library entries are embedded without a prefix. Cosine scores are rounded to two decimal places, with lexical statement Jaccard breaking ties. The rank excludes the symbol and type signature channel because it saturates at 1.0 on this corpus. Retrieval is deterministic, so each run receives the same K library entries for a given theorem.

F.6 Base model, serving and decoding

The base model is Qwen3.5-397B-A17B in FP8 (Qwen Team, 2026a), the same open-weight model used in the test-time scaling study of Appendix C, but served differently here. We serve it with SGLang rather than vLLM, at tensor-parallel size 8 rather than 4. Each replica uses eight B200 GPUs, a context window of 65,536 tokens and GPU memory utilization of 0.92. A client distributes requests across 16 replicas and retries transport errors on the next live replica. The model runs in its default thinking mode without a chat-template argument. Library construction and all evaluation conditions use one decoding profile pinned by hash: temperature 0.6, top-p 0.95, top-k 20, min-p 0.0, presence penalty 0.0, repetition penalty 1.0, and a client-side limit of 32,768 output tokens. We do not pin a decode seed. The three runs are independent samples from the decoder, and the reported standard deviation measures variation across these runs.

F.7 Prompt

All three conditions use one template containing a system prefix, the target, base premises, worked examples and previous attempts. Only the worked-examples section changes. Zero-shot renders this section as `None available.` rather than omitting it. Base premises and previous attempts are empty in every condition, so the schema library is the only source of retrieved proof knowledge. Each schema is rendered as a compilable Lean `example` with its theorem statement and the proof body accepted by Lean.

Prompt 2: ProofEvolve prompt with relevant schemas

````
[system]
You are proving a theorem in Lean 4 with Mathlib.

Reply with a single fenced ```lean block containing ONLY the tactic block
that completes the given theorem. Do not restate the theorem, do not add
imports, and do not declare anything at the top level: the surrounding file
is fixed and your reply is inserted after ':= by'.

If you cannot close the goal, still reply with your best tactic block.
Never emit 'sorry'.

[user]
## Target
The file below is frozen. Your tactic block is inserted where marked.
```lean
import Mathlib

theorem <name> <statement> := by
  <your tactic block here>
```
## Proof state
One open obligation: the theorem statement above.
## Base premises
None selected for this request.
## Worked examples
Solved problems from your own library, for reference. They are different
problems; adapt the techniques, do not copy.
### Example 1
```lean
example <library statement 1> := by
<verified proof body 1>
```
... (K examples)
## Previous attempts
None.
````

F.8 Conditions, budget and compute

Every condition gives one attempt per theorem with no self-repair or second sample. The Lean environment imports Mathlib. Relevant retrieval uses $K \in \{8, 16, 32, 64\}$ over the full library. We also evaluate library sizes {1,000, 2,000, 4,000, full} at K=8. Random retrieval uses K=8 with a fixed draw seed for each run. We run all nine configurations three times over the same 1,000 evaluation candidates, then score the common 744-theorem screened subset in every condition. Every run produces one record per candidate. Condition contrasts are computed as paired differences across matched runs.

For the copy rate in Section 5.6, we remove comments and normalize whitespace in every accepted proof body. We then compare it with the eight proof bodies shown for that theorem. For each run index, we count cases where relevant retrieval closes a theorem and the corresponding zero-shot run leaves it open.

The fleet contains 16 FP8 replicas of 8 B200 GPUs each, or 128 B200 GPUs in total. Building the library over 9,968 source theorems takes about five hours of fleet time. A three-run sweep of one configuration takes about seven hours. The complete study uses a few thousand B200-GPU-hours across library construction, the depth and size sweeps and leakage screening.

G Retrieval-to-Proof Traces on Lean Workbook

Section 5.6 reports that 322 of the 351 theorems closed by relevant retrieval and left open by zero-shot, or 91.7%, do not reproduce any shown proof verbatim. That is an aggregate. This appendix gives the individual form of it: two complete traces from the evaluation theorem, through what the retriever actually returned, to

Condition	Solve rate (%)	Lift (pp)
Zero-shot	49.5 ± 0.7	0.0
Random retrieval (K=8)	49.6 ± 1.0	+0.1
Relevant retrieval, depth (full library)		
K=8	53.4 ± 0.7	+3.9
K=16	54.1 ± 0.9	+4.6
K=32	52.9 ± 0.9	+3.4
K=64	54.8 ± 0.5	+5.3
Relevant retrieval, library size (K=8)		
1,000 proofs	52.8 ± 0.3	+3.3
2,000 proofs	52.2 ± 0.8	+2.6
4,000 proofs	53.7 ± 0.3	+4.2
Full library (5,546)	53.4 ± 0.7	+3.9

Table 7 Solve rate on the 744 screened evaluation theorems. We report the mean over three independent runs, and the $\pm$ figure is the run-to-run standard deviation *of the solve rate*. Lift is the paired difference from zero-shot, computed per run before rounding, so it need not equal the difference of the two rounded means.

the proof Lean accepted. Each trace runs through five stages. **(1)** the evaluation theorem and why it is not immediate; **(2)** two of the eight schemas the retriever placed in the proposal context; **(3)** the proof the model produced and the kernel accepted; **(4)** a comparison with the reference proof released with Lean Workbook, which shows the two take different routes; and **(5)** the outcome of all three conditions over the three runs. The selected Lean statements preserve the informal problem and avoid truncated natural-number arithmetic, inconsistent assumptions and trivial goals.

Example 1. A bound on $[0, 1]$

(1) Target. For $0 \le x \le 1$, show that $|x(x-1)(x^6+2x^4+3x^2+4)| < 5/2$. The two natural factor bounds do not prove the strict inequality: multiplying $x(1-x) \le 1/4$ and $x^6+2x^4+3x^2+4 \le 10$ gives only $\le 5/2$.

```
(x : ℝ) (hx : 0 ≤ x ∧ x ≤ 1) :
  |x * (x - 1) * (x^6 + 2 * x^4 + 3 * x^2 + 4)| < 5 / 2
```

(2) Retrieved. Two of the eight schemas placed in the proposal context. Both are bounds on the same interval, and neither states the target.

```
example (x : ℝ) (hx : 0 ≤ x ∧ x ≤ 1) :
    0 ≤ x - x^2 ∧ x - x^2 ≤ 1/4 := by
  cases hx
  constructor <;> nlinarith [sq_nonneg (x - 1/2)]

example (x : ℝ) (hx : 0 ≤ x ∧ x ≤ 1) :
    x * (x ^ 3 - 6 * x + 9) ≤ 4 := by
  nlinarith [sq_nonneg (x - 1), sq_nonneg (x^2 - 1), sq_nonneg (x^2 - x),
    mul_nonneg hx.1 (sq_nonneg (x - 1)),
    mul_nonneg (sub_nonneg.mpr hx.2) (sq_nonneg (x - 1))]
```

(3) Generated. The kernel-accepted proof. It determines the sign of the product first, which neither retrieved schema does, and then reuses the sq_nonneg$(x - 1/2)$ hint that both of them turn on.

```
cases hx with
| intro hx1 hx2 =>
  have h1 : x * (x - 1) ≤ 0 := by nlinarith
  have h2 : x^6 + 2 * x^4 + 3 * x^2 + 4 ≥ 4 := by
    nlinarith [pow_nonneg hx1 2, pow_nonneg hx1 4, pow_nonneg hx1 6]
  have h3 : x * (x - 1) * (x^6 + 2*x^4 + 3*x^2 + 4) ≤ 0 := by
    nlinarith [pow_nonneg hx1 2, pow_nonneg hx1 4, pow_nonneg hx1 6]
  rw [abs_of_nonpos h3]
  nlinarith [sq_nonneg (x - 1/2), pow_nonneg hx1 2, pow_nonneg hx1 3,
    pow_nonneg hx1 4, pow_nonneg hx1 5, pow_nonneg hx1 6,
    mul_nonneg hx1 (sq_nonneg (x - 1/2)),
```

Example 1. A bound on $[0, 1]$ (continued)

```
    mul_nonneg (sub_nonneg.mpr hx2) (sq_nonneg (x - 1/2))]
```

(4) Compared. The reference proof released with Lean Workbook splits the absolute value into two inequalities without determining the sign of the product. None of its auxiliary inequalities appears in the accepted proof.

```
rw [abs_lt]
constructor <;>
  nlinarith [pow_nonneg (sub_nonneg.mpr hx.1) 0, ...,
          pow_nonneg (sub_nonneg.mpr hx.1) 9]
```

(5) Outcome over three runs. Zero-shot closes zero runs, relevant retrieval closes two and random retrieval closes one.

Example 2. A three-part symmetric conclusion

(1) Target. If $x, y, z > 0$ and $x^4 + y^4 + z^4 = 3$, then $x + y + z \leq 3$, $xy + yz + zx \leq 3$ and $xyz \leq 1$. The conclusion is a conjunction, so a single inequality tactic has to discharge three different bounds at once.

```
(x y z : ℝ) (hx : 0 < x) (hy : 0 < y) (hz : 0 < z)
    (h : x^4 + y^4 + z^4 = 3) :
  x + y + z ≤ 3 ∧ x*y + y*z + z*x ≤ 3 ∧ x*y*z ≤ 1
```

(2) Retrieved. Two of the eight schemas. Both are symmetric three-variable inequalities under a different constraint.

```
example (x y z : ℝ) (hx : 0 ≤ x) (hy : 0 ≤ y) (hz : 0 ≤ z)
    (h : x^2 + y^2 + z^2 + x*y*z = 4) :
    3*(x^2*y + y^2*z + z^2*x) ≤ (x + y + z)*(x^2 + y^2 + z^2) := by
  nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
    mul_nonneg hx hy, mul_nonneg hy hz, mul_nonneg hz hx,
    mul_nonneg (sq_nonneg (x - y)) hz, ...]

example (x y z : ℝ) (hx : 0 ≤ x ∧ x ≤ 1) (hy : 0 ≤ y ∧ y ≤ 1)
    (hz : 0 ≤ z ∧ z ≤ 1) :
    2 * (x*y + y*z + z*x) ≤ 3*x*y*z + x + y + z := by
  have h1 : 0 ≤ (1 - x) * (1 - y) * z := by
    apply mul_nonneg
    · apply mul_nonneg <;> linarith
    · linarith
  ...
```

(3) Generated. The kernel-accepted proof builds eleven named intermediate results and assembles the conjunction at the end, rather than attacking it in one step. Supporting expressions are elided at ... for space.

```
have h1 : x^2 + y^2 + z^2 ≤ 3 := by
  nlinarith [sq_nonneg (x^2 - 1), ..., h]
have h2 : x*y + y*z + z*x ≤ x^2 + y^2 + z^2 := by
  nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x)]
have h3 : x*y + y*z + z*x ≤ 3 := by linarith [h1, h2]
have h4 : (x + y + z)^2 ≤ 9 := by nlinarith [h1, h2]
have h5 : x + y + z ≤ 3 := by nlinarith [sq_nonneg (x + y + z - 3), h4]
have h6 : x^2*y^2 + y^2*z^2 + z^2*x^2 ≤ 3 := by
  nlinarith [sq_nonneg (x^2 - y^2), ..., h]
have h7 : (x^2 + y^2 + z^2) * (x^2*y^2 + y^2*z^2 + z^2*x^2)
    ≥ 9 * x^2 * y^2 * z^2 := by
  nlinarith [mul_nonneg (sq_nonneg x) (sq_nonneg (y^2 - z^2)), ...]
have hP : 0 ≤ x^2*y^2 + y^2*z^2 + z^2*x^2 := by
  nlinarith [sq_nonneg (x*y), sq_nonneg (y*z), sq_nonneg (z*x)]
have h_SP : (x^2 + y^2 + z^2) * (x^2*y^2 + y^2*z^2 + z^2*x^2) ≤ 9 := by
  nlinarith [mul_nonneg hP (sub_nonneg.mpr h1), ...]
have h8 : x^2 * y^2 * z^2 ≤ 1 := by nlinarith [h7, h_SP]
have h9 : x * y * z ≤ 1 := by nlinarith [sq_nonneg (x*y*z - 1), h8]
exact ⟨h5, h3, h9⟩
```

(4) Compared. The released reference proof splits the conjunction first and discharges all three parts with one shared list of auxiliary inequalities. Six of its nine inequalities appear nowhere in the accepted proof.

```
refine' ⟨_, _, _⟩
all_goals nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
  sq_nonneg (x + y), sq_nonneg (y + z), sq_nonneg (z + x), h,
  sq_nonneg (x - 1), sq_nonneg (y - 1), sq_nonneg (z - 1)]
```

(5) Outcome over three runs. Zero-shot closes zero runs, relevant retrieval closes two and random retrieval closes one.