TARGET-DRIVEN AUTOMATED CONJECTURING FOR NEURAL THEOREM PROVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Specialised neural theorem proving systems are commonly trained through reinforcement learning on large collections of formal problems, often autoformalised from human-written sources. This dependence on human-curated data limits their scalability and prevents theorem-proving agents from improving autonomously through mathematical exploration, since each new training curriculum must ultimately be supplied by humans. To address this challenge, we introduce **TACO** (**T**arget-Driven **A**utomated **Co**njecturing), a framework that constructs theorem-proving curricula from model-generated conjectures. Since individual conjectures have small and noisy training effects, TACO evaluates them collectively: starting from a small seed set, it generates candidate conjecture sets, trains Prover variants on them, and scores each set by the improvement its corresponding Prover achieves on a fixed target set. This evolutionary procedure builds the curriculum stage by stage while iteratively updating the seeds and base Prover. Across four mathematical domains, using held-out problems selected from publicly available Lean datasets, TACO increases the number of held-out test problems solved within 32 attempts from $10/128$ to $43/128$ after ten iterations, while successful proof attempts increase from $16/4096$ to $358/4096$. These results demonstrate that model-generated conjectures can provide effective training curricula, enabling theorem proving systems to improve through targeted mathematical exploration while reducing reliance on human-written problems.

1 INTRODUCTION

Recent years have seen rapid progress in specialised neural theorem provers trained to produce formally verified proofs, with systems approaching saturation on miniF2F (Zheng et al., 2022; Hubert et al., 2025) and achieving substantial progress on more challenging formal benchmarks such as PutnamBench (Chen et al., 2025). Specialised neural theorem proving models are trained through reinforcement learning pipelines supplied with hundreds of thousands of human-written problems (Hubert et al., 2025; Wang et al., 2025; Ren et al., 2025), often translated into formal languages through autoformalisation (Wu et al., 2022). This dependence on human-curated data constrains progress by the availability, coverage, and quality of existing problem collections. More fundamentally, each new training curriculum must ultimately be supplied by humans, preventing theorem-proving agents from improving autonomously through mathematical exploration. One way to reduce this dependence is to construct training curricula from model-generated formal conjectures.

Recent work has begun to address this human-curated curriculum bottleneck by generating synthetic formal problems for theorem-prover training. Early systems such as MetaGen (Wang & Deng, 2020) generate new formal statements to expand prover training data, while more recent approaches construct iterative or target-specific curricula from generated conjectures or theorem variants (Hubert et al., 2025; Poesia et al., 2024; Dong & Ma, 2025; Bailey et al., 2026). Together, these works establish model-generated formal problems as a viable source of training curricula for theorem provers.

A key question is which generated conjectures should be selected for training. Existing methods use intrinsic or heuristic criteria: Minimo favours hard but provable conjectures (Poesia et al., 2024), Self-Play Theorem Prover selects conjectures that are barely provable and satisfy elegance criteria (Dong & Ma, 2025), and Scaling Self-Play with Self-Guidance additionally scores their relevance to unsolved targets and how clean and natural they appear (Bailey et al., 2026). These criteria may

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

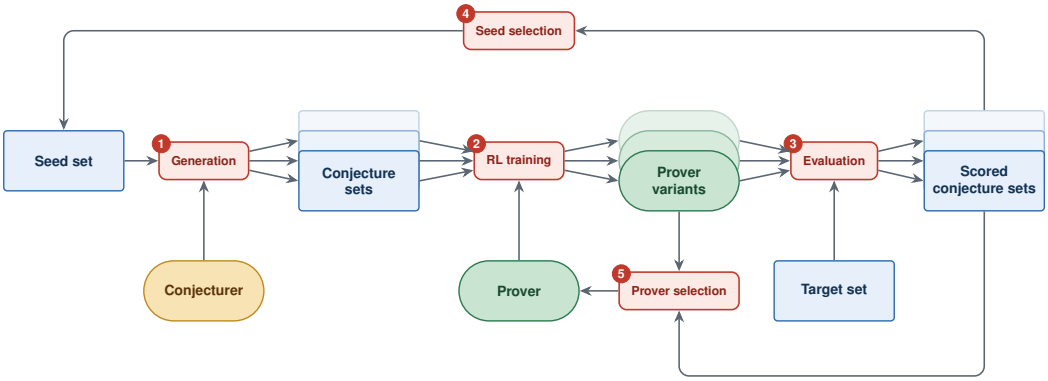


Figure 1: **One iteration of TACO.** (1) For each seed theorem, the Conjecturer generates a set of conjectures similar to the seed, providing a candidate next stage of training. (2) A separate copy of the base Prover is trained on each conjecture set using reinforcement learning. (3) The Prover variants are evaluated on a fixed target set, and each conjecture set is scored by the improvement achieved by its corresponding variant. (4) Successfully proved conjectures are selected from the highest-scoring sets for the next iteration. (5) The best-performing Prover variant becomes the new base Prover. TACO thus guides conjecture selection by the measured training utility of generated sets.

correlate with downstream training utility, but they do not directly measure whether training on the selected conjectures improves the prover.

A central challenge is that the downstream utility of a conjecture can only be directly established by training a prover on it and measuring the resulting change in proving performance. Doing this separately for every conjecture would be prohibitively expensive, while the marginal effect of an individual conjecture within a training batch is typically small and noisy. Existing methods therefore rely on proxy criteria, such as perceived difficulty, relevance, elegance or provability (Poesia et al., 2024; Dong & Ma, 2025; Bailey et al., 2026), which may correlate with downstream utility, but do not measure it directly.

The open problem is therefore how to determine which model-generated conjectures are useful for training. In particular, can conjecture selection be driven by the measured improvement induced in a prover, rather than by proxy criteria such as difficulty, novelty or elegance?

We introduce **TACO**, **T**arget-Driven **A**utomated **Co**njecturing (Figure 1), a framework that selects conjectures by directly measuring the improvement they induce in downstream proving performance. Since the marginal utility of an individual conjecture is too small and noisy to estimate reliably, TACO evaluates conjectures at the level of generated sets: each set represents one possible next stage of training and is scored by the improvement achieved by a Prover trained on it. TACO follows an evolutionary procedure that alternates between conjecture generation and Prover training, iteratively updating both the conjecture curriculum and the base Prover.

Across four mathematical domains, TACO increases the total number of held-out test problems solved within 32 attempts from 10/128 to 43/128 after ten iterations, while also substantially increasing the number of successful proof attempts. These results demonstrate that model-generated conjectures can form effective training curricula for neural theorem provers, reducing reliance on additional human-written problems during curriculum construction and providing a step towards more autonomous mathematical exploration. Our contributions can therefore be summarised as:

1. **TACO**, an evolutionary framework that constructs theorem-proving curricula from a small initial seed set while iteratively updating both the seeds and the base Prover.
2. A target-driven evaluation procedure that scores each conjecture set by the downstream Prover improvement it induces rather than by intrinsic proxy criteria.
3. An empirical demonstration that iterative training on model-generated conjectures improves theorem-proving performance on held-out competition-style problems.

2 Related Work

Reinforcement learning and autoformalisation for formal theorem proving. Specialised theorem-proving models increasingly rely on reinforcement learning, using proof-assistant verification as a training signal to improve proof generation. Polu et al. (2023) showed that reinforcement learning through expert iteration can solve increasingly difficult formal problems without requiring reference proofs, but the theorem statements must still be supplied externally. Subsequent systems have scaled reinforcement learning to larger models and problem collections, producing major improvements on Lean benchmarks and competition-level mathematics (Xin et al., 2025; Wu et al., 2025; Wang et al., 2025; Ren et al., 2025; Hubert et al., 2025). Autoformalisation (Wu et al., 2022) expands the supply of formal training problems by translating human-written mathematics into proof-assistant languages (Xin et al., 2024; Hubert et al., 2025; Wang et al., 2025; Ren et al., 2025), changes the representation of existing problems rather than generating new statements, and remains limited by the available formal language and library. Training curricula therefore remain constrained by the quantity, coverage, and difficulty distribution of available human-written problems, motivating approaches that generate new training statements.

Model-generated conjectures. Beyond translating existing problems, several systems generate new formal conjectures or theorem variants for prover training. MetaGen learns to synthesise theorems and proofs in Metamath, with its generator trained by imitation of human proofs or rewards encouraging similarity to human-written statements (Wang & Deng, 2020). However, conjecture usefulness is judged through resemblance to human data rather than downstream prover improvement. AlphaProof generates large problem-specific curricula of synthetic variants, including simplifications, generalisations, and proposed lemmas, which are validated, expanded evolutionarily, deduplicated, and pooled for specialist training (Hubert et al., 2025), but does not select candidate curricula according to their realised training effect. Minimo jointly trains a system to generate hard but provable conjectures and to prove them from axioms (Poesia et al., 2024), using intrinsic difficulty rather than improvement on a separate proving objective. Self-Play Theorem Prover extends iterative conjecturing and proving to Lean and Isabelle (Dong & Ma, 2025), but selects conjectures that are barely provable and satisfy heuristic elegance criteria rather than according to their measured downstream utility.

Selecting useful conjectures. More recent work has focused explicitly on selecting conjectures that are useful for subsequent proving. Scaling Self-Play with Self-Guidance addresses the tendency of difficulty-based self-play to generate artificially complex problems by introducing a Guide that scores conjectures according to their relevance to unsolved targets and how clean and natural they appear (Bailey et al., 2026). UseFor instead measures usefulness through reusability in subsequent proofs and contributions to increasing proof likelihood (Kasriel et al., 2025). These methods move beyond purely intrinsic criteria, but still estimate conjecture usefulness without measuring the improvement produced by training a prover on the selected conjectures. More broadly, Learning to Teach learns data-selection strategies from feedback on student learning (Fan et al., 2018), providing a precedent for using learning outcomes to guide curriculum construction. TACO brings this principle to neural theorem proving by selecting candidate conjecture sets according to the realised improvement achieved by Provers trained on them, rather than by an intrinsic or predicted notion of conjecture usefulness.

3 TACO: Target-Driven Automated Conjecturing

TACO (Figure 1) is a model-agnostic iterative evolutionary procedure for constructing theorem-proving curricula from model-generated conjectures. The framework can be instantiated with any Conjecturer model capable of generating formal conjectures from input seed problems, and any Prover model that can be trained on formal theorem-proving tasks and evaluated on a target set.

The algorithm is initialised with a base Prover $P^{(0)}$, a Conjecturer C, an initial seed set:

$$\mathcal{S}^{(0)} = \{x_1, \ldots, x_s\},$$

and a fixed target set $\mathcal{T}_{\text{target}}$ containing t problems. A corresponding test set $\mathcal{T}_{\text{test}}$, drawn from the same distribution, is held out from the entire TACO loop.

The seed set determines the regions of conjecture space explored by the Conjecturer, while the target set specifies the proving capabilities that guide curriculum construction.

At each iteration i, TACO performs the following steps:

1. **Conjecture generation.** (Figure 1①) For each seed problem $x_j \in \mathcal{S}^{(i)}$, the Conjecturer generates a set of formal conjectures $\mathcal{C}_j^{(i)}$. This yields one candidate training set per seed problem.

2. **Prover training.** (Figure 1②) For each candidate training set $\mathcal{C}_j^{(i)}$, a separate copy of the current Prover $P^{(i)}$ is trained on its conjectures using Group Relative Policy Optimization (GRPO) (Shao et al., 2024), with rewards based on proof correctness as checked by the proof assistant, producing a trained Prover variant $P_j^{(i)}$. We also record the subset $\widehat{\mathcal{C}}_j^{(i)}$ of conjectures proved in some but not all of their rollouts, i.e. those that provide a non-zero training signal. Only these conjectures are eligible to become seeds for the next iteration, ensuring that every retained seed has a proof accepted by the underlying proof assistant.

3. **Target-set evaluation.** (Figure 1③) Each trained Prover variant and the current Prover are evaluated on the fixed target set using k proof attempts per problem. For a Prover P, let $c_x(P)$ denote the number of correct proof attempts on target problem x. We define:

$$\mathrm{Perf}(P; \mathcal{T}_{\mathrm{target}}) = \sum_{x \in \mathcal{T}_{\mathrm{target}}} \left[1 - \left(1 - \frac{c_x(P)}{k} \right)^m \right],$$

   where m is a fixed saturation parameter. This metric estimates the number of target problems solved at least once in m attempts, thereby rewarding broad coverage while assigning diminishing returns to repeated successes on the same problem. The score assigned to each candidate conjecture set is its improvement over the current Prover:

$$\mathrm{Score}(\mathcal{C}_j^{(i)}; \mathcal{T}_{\mathrm{target}}) = \mathrm{Perf}(P_j^{(i)}; \mathcal{T}_{\mathrm{target}}) - \mathrm{Perf}(P^{(i)}; \mathcal{T}_{\mathrm{target}}).$$

   Subtracting the shared base-Prover score leaves candidate rankings unchanged, but determines whether the Prover is updated. Scores reflect set-level utility under the current Prover and training budget and remain subject to training and evaluation variability.

4. **Seed selection.** (Figure 1④) Candidate sets scoring above the mean candidate score in the current iteration are allocated positions in the next seed set in proportion to their excess over this mean. Within each selected set, the allocated number of proved conjectures is sampled uniformly without replacement. This relative selection rule applies even when all trained variants underperform the current Prover, allowing conjecture-space exploration to continue. Ties, rounding, and sets with too few eligible conjectures are handled as specified in Appendix D.

5. **Base Prover update.** (Figure 1⑤) If the highest-scoring trained Prover variant achieves a positive score, it becomes the base Prover for the next iteration. Otherwise, the current Prover is retained unchanged.

Balancing coverage and reliability. Optimising pass rate alone can favour repeatedly solving the same problems over expanding coverage, since every additional success receives equal credit. Conversely, observed pass@k gives no additional credit for repeated successes on a problem. Perf interpolates between these two extremes: at $m = 1$, it is proportional to pass rate; as $m \to \infty$, it approaches the number of problems with at least one observed success. For finite $m > 1$, each additional success on a problem receives positive but diminishing credit, so first successes on rarely solved problems count most while consolidation is still rewarded.

This evolutionary loop implements target-driven search over candidate training sets with set-level credit assignment. TACO prioritises conjectures from sets whose trained Provers achieve higher measured target performance rather than selecting them according to intrinsic proxy criteria. The following section describes the models, training procedure, data construction and experimental protocol.

4 EXPERIMENTS AND EVALUATION

We evaluate TACO through four research questions:

RQ1. Can iterative training on model-generated conjectures expand the set of held-out test theorems a Prover can solve?

RQ2. Can iterative training on model-generated conjectures increase the pass rate on held-out test theorems?

RQ3. How does this training change the use of domain-relevant lemmas and general-purpose automation in proof attempts?

RQ4. How do initial seed alignment and iterative seed selection affect held-out performance?

4.1 IMPLEMENTATION

Models. We instantiate the Prover with Qwen3.5-2B, fine-tuned on Lean statement-proof pairs to generate proofs from formal statements. For the Conjecturer, we use Qwen3.8-27B with FP8 quantisation, without additional fine-tuning. Its weights remain fixed throughout the TACO loop: conjecture generation adapts through the evolving seeds. We use open-weight Qwen models because TACO requires training many Prover copies, which the small 2B model makes affordable, and a fixed, reproducible Conjecturer, for which the 27B model follows detailed instructions closely enough to control conjecture generation through prompting alone.

Training data. We combine NuminaMath-LEAN (Wang et al., 2025) and Lean-Workbook-Plus (Ying et al., 2024), retaining only examples with available proof. We normalise theorem statements by anonymising theorem names and removing comments, unify imports, and retain examples that compile successfully in Lean 4.15. We deduplicate the resulting corpus by exact normalised statement and divide it into a 75–25 training–evaluation split. We then augment the evaluation split with problems from the Pythagoras-LM SFT dataset (Leang et al., 2026), removing statements that appear in either split after the same normalisation, and retaining those verified in Lean. The two splits are therefore disjoint by normalised statement. The training split supplies the initial Prover fine-tuning data, while the augmented evaluation pool supplies the seed, target, and test sets.

Initial Prover training. We initialise the Prover through supervised fine-tuning on 16,673 statement-proof pairs of the training split, with the objective of generating the corresponding proof given a formal theorem statement. We train for one epoch, comprising about 1,040 optimisation steps, with a learning rate of 10^{-6} and a batch size of 16. The resulting model serves as the initial Prover $P^{(0)}$. Details of GRPO training, reward computation, and Lean verification are given in Appendix E.

Conjecture generation. Given a seed statement, each generation call prompts the Conjecturer to produce eight standalone Lean statements without proofs. We use three prompts controlling proximity to the seed: *close* variants preserve its structure with small modifications; *intermediate* variants introduce an additional difficulty; and *exploratory* variants explore other problem families within the seed's domain. The full prompts are given in Figure 4 (Appendix E).The prompts encourage modest increases in difficulty and include an input-output example illustrating the expected format and degree of variation. We remove duplicates within each candidate set. Generated conjectures are also not systematically checked against target or test statements; such overlap is unlikely and was not observed in our experiments, but cannot be ruled out.

Across repeated calls for each seed, we aim to collect 64 conjectures: 32 close, 16 intermediate, and 16 exploratory variants. This procedure does not detect general mathematical equivalence. Generation details are given in Appendix E.

4.2 EXPERIMENTAL SETUP

Evaluation data. Our evaluation pool contains 26,418 unique theorem statements from NuminaMath-LEAN, Lean-Workbook-Plus, and Pythagoras-LM, constructed as described in Sec-

tion 4.1. Our main evaluation focuses on four mathematical domains: radical algebra, geometry of areas, geometry of volumes, and trigonometry.

Seed, target, and test selection. Seed, target, and test sets are constructed from the augmented evaluation split. For each mathematical domain, we first define a candidate pool by retaining theorem statements that contain at least one identifier from a domain-specific keyword list. For example, the trigonometry pool is defined using `Real.sin`, `Real.cos`, `Real.tan`, `Real.arcsin`, `Real.arccos`, and `Real.arctan`. We then evaluate the initial Prover $P^{(0)}$ on every candidate theorem using 16 proof attempts. The 16 initial seeds are sampled uniformly without replacement from the theorems solved in at least 6 of these 16 attempts. The target and test sets, each containing 32 theorems, are sampled without replacement from the theorems solved in at most one attempt, with priority given to theorems that are never solved. The resulting target and test sets are disjoint. Target and test screening deliberately selects problems that the initial Prover rarely solves; the results therefore characterise improvement on a difficult subset. Full details, including domain filters, are given in Appendix C.

Protocol and evaluation. We run TACO for 10 iterations, generating up to 64 conjectures per seed and selecting up to 16 seeds for the next iteration. Each candidate and current base Prover is evaluated on the target set using 32 proof attempts per theorem. Conjecture-set scoring, seed selection, and Prover updates use the performance metric Perf defined in Section 3, with saturation exponent $m = 12$, chosen through limited preliminary exploration on the target sets only and fixed across all reported experiments. This choice provides a practical balance between rewarding new successes and consolidating existing ones; we do not claim it is optimal or conduct a systematic sensitivity analysis. We report held-out test performance at iterations 0, 5, and 10, using 32 proof attempts per theorem. The tables compare initial and final performance, while Figure 2 also includes iteration five. Pass@32 is the fraction of test theorems solved at least once; pass rate is the fraction of successful proof attempts out of 1,024 per domain. Test results are used only for reporting and do not guide the loop.

Ablations. For each domain, we compare four conditions using the same target and test sets: standard TACO, random initial seeds only, random iterative seed selection only, and both randomisations combined. Random initial seeds are sampled uniformly without replacement from the full evaluation pool, irrespective of domain, among theorems solved in at least 6 of 16 attempts by the initial Prover. Random iterative selection replaces the target-based allocation with uniform sampling without replacement from the pooled eligible conjectures. All four conditions retain the same target-based Prover-update rule.

Infrastructure. All experiments use NVIDIA GH200 GPUs for model serving and training. Generation uses vLLM, GRPO training uses TRL, and Lean interaction is handled through the Kimina Lean Server (Dos Santos et al., 2025). Compute costs are reported in Appendix E.

4.3 Evaluation

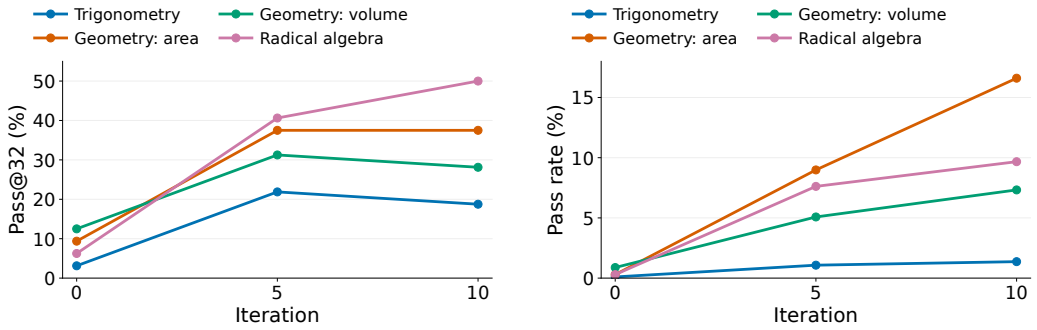
Table 1 reports Prover performance on the held-out test sets before TACO training and after ten iterations. Pass@32 denotes the fraction of test theorems solved at least once in 32 attempts, while pass rate denotes the fraction of successful proof attempts across all 1024 attempts on the 32 test theorems.

RQ1: TACO improves held-out theorem proving. After ten iterations, the aggregate number of test theorems solved at least once in 32 attempts increases from $10/128$ to $43/128$ (7.8% to 33.6%), corresponding to a 4.3-fold increase under the same evaluation sampling budget (Table 1). The net gains are positive in every domain: 14 additional theorems in radical algebra, nine in geometry of areas, five in geometry of volumes, and five in trigonometry. Because test results do not guide curriculum selection or Prover updates, these gains provide evidence that improvements extend beyond the target problems used to guide selection.

Between iterations five and ten, aggregate coverage changes little ($42/128$ to $43/128$), while successful proof attempts increase from $233/4096$ to $358/4096$ (Figure 2). Later training therefore improves overall proof-generation reliability more than it expands the set of solvable problems.

Table 1: Held-out test performance before and after ten TACO iterations. Base denotes the initial Prover, and Iter. 10 denotes the retained Prover after ten iterations. Pass@32 reports the fraction of theorems solved at least once in 32 attempts, and pass rate reports the fraction of successful proof attempts. Both theorem coverage and pass rate increase in all four domains.

	Base		Iter. 10	
Domain	**Pass@32**	**Pass rate**	**Pass@32**	**Pass rate**
Radical algebra	2/32	3/1024	**16/32**	**99/1024**
Trigonometry	1/32	1/1024	**6/32**	**14/1024**
Geometry: area	3/32	3/1024	**12/32**	**170/1024**
Geometry: volume	4/32	9/1024	**9/32**	**75/1024**
Total	10/128	16/4096	**43/128**	**358/4096**



(a) Pass@32 (b) Pass rate

Figure 2: Held-out test performance of TACO at iterations 0, 5, and 10 across four mathematical domains. Pass@32 and pass rate are expressed as percentages of 32 theorems and 1,024 proof attempts per domain, respectively. Both metrics improve from initial to final evaluation in every domain; between iterations five and ten, aggregate coverage changes little while pass rate continues to increase.

RQ2: TACO improves proof-generation reliability. Across the four domains, successful test proof attempts increase from 16/4096 to 358/4096 (Table 1). This improvement reflects more than broader theorem coverage: on the fixed subset of ten theorems solved by the initial Prover, successful attempts increase from 16/320 to 148/320 (5.0% to 46.3%). Nine of these ten theorems receive more successful proofs, and all remain solvable at iteration ten.

Together, RQ1 and RQ2 show improvements in both coverage and reliability. A plausible explanation is that generated conjectures provide repeated practice of proof procedures that also apply to held-out problems. Reinforcing these procedures can make existing successes more frequent and previously unsuccessful problems accessible within the evaluation budget. Appendix B illustrates this: in radical algebra, generated conjectures exercise square-root bounds proved by supplying nonnegative squares to `nlinarith`, and successful attempts on a related target problem increase from 0/32 to 17/32. The increased references to existing proof tools examined in RQ3 are also consistent with this interpretation.

RQ3: Training increases explicit use of domain-relevant lemmas. Inspection of all test proof attempts, successful or not, reveals more frequent references to domain-relevant lemmas after training. For example, the fraction of attempts mentioning `sqrt_nonneg` increases from 14.3% to 79.6% in radical algebra, and `pi_pos` from 0.4% to 48.8% in geometry of volumes (Table 3). General-purpose automation also becomes more frequent: `field_simp` increases from 6.9% to 85.1% in geometry of areas, `linarith` from 18.8% to 96.3% in geometry of volumes, and `nlinarith` from 21.6% to 56.5% in trigonometry.

Table 2: Effect of initial seed alignment and iterative seed selection on held-out test performance. Base denotes the shared evaluation of the initial Prover, and Iter. 10 reports each configuration's Prover after ten iterations. Pass@32 and pass rate are defined in Table 1. Target-aligned initial seeds come from the target domain; random initial seeds have no domain restriction. Iterative selection is either target-driven or uniform over eligible conjectures. All configurations within each domain share the same target and test sets and retain target-based best-Prover selection. With target-driven iterative selection, aligned initial seeds yield higher pass rates in all four domains; the benefit of target-driven over random iterative selection varies by domain.

			Base		Iter. 10	
Domain	**Initial seeds**	**Iterative selection**	**Pass@32**	**Pass rate**	**Pass@32**	**Pass rate**
Radical algebra	Target-aligned	Target-driven	$2/32$	$3/1024$	$\mathbf{16/32}$	$\mathbf{99/1024}$
	Random	Target-driven			$11/32$	$47/1024$
	Target-aligned	Random			$9/32$	$42/1024$
	Random	Random			$9/32$	$42/1024$
Trigonometry	Target-aligned	Target-driven	$1/32$	$1/1024$	$\mathbf{6/32}$	$14/1024$
	Random	Target-driven			$1/32$	$2/1024$
	Target-aligned	Random			$5/32$	$\mathbf{39/1024}$
	Random	Random			$4/32$	$22/1024$
Geometry: area	Target-aligned	Target-driven	$3/32$	$3/1024$	$12/32$	$\mathbf{170/1024}$
	Random	Target-driven			$11/32$	$70/1024$
	Target-aligned	Random			$13/32$	$92/1024$
	Random	Random			$\mathbf{14/32}$	$169/1024$
Geometry: volume	Target-aligned	Target-driven	$4/32$	$9/1024$	$9/32$	$75/1024$
	Random	Target-driven			$9/32$	$49/1024$
	Target-aligned	Random			$\mathbf{11/32}$	$\mathbf{138/1024}$
	Random	Random			$10/32$	$79/1024$

These observations suggest a possible interaction between domain-relevant lemmas and general-purpose automation. For example, `sqrt_nonneg` and `pi_pos` supply sign information that arithmetic tactics can exploit. Training on generated conjectures may therefore reinforce the invocation of mathematical facts that prepare a goal for automated reasoning, rather than merely encouraging more frequent calls to automation. Most of these lemmas already occur in the initial Prover's outputs, so the evidence supports increased references to existing tools rather than establishing acquisition of new mathematical knowledge. Since these counts include unsuccessful attempts, they do not establish that the lemmas were applied correctly or contributed to completed proofs.

RQ4: Seed alignment benefits target-driven runs, while selection effects vary across domains. Table 2 compares target-aligned and random initial seeds under target-driven and random iterative seed selection, as defined in the experimental setup.

With target-driven selection, aligned initial seeds achieve higher pass rates in all four domains and higher coverage in three, tying in geometry of volumes (aggregate pass@32 $43/128$ versus $32/128$; pass rate $358/4096$ versus $168/4096$). Under random iterative selection, this advantage disappears ($38/128$ versus $37/128$; $311/4096$ versus $312/4096$). With aligned seeds, target-driven selection outperforms random selection on both metrics only in radical algebra; results are mixed in trigonometry and geometry of areas, and random selection performs better in geometry of volumes. Selecting seeds by measured training utility can therefore help, but does not uniformly outperform random selection.

Tactic and lemma usage (Appendix A) helps explain the alignment effect: aligned seeds yield more frequent references to domain-relevant lemmas than random seeds in 19 of the 24 lemma–configuration comparisons (Table 3). In trigonometry, with random seeds and target-driven selection, the use of `nlinarith` rises from 21.6% to 81.6% of test attempts while trigonometric identities remain rare (`cos_sq`: 3.2%, versus 33.2% with aligned seeds), and pass rate barely changes:

seed selection can only choose among the conjecture families that the seeds make available. Lemma usage does not, however, track the effect of iterative selection: in geometry of volumes, target-driven selection yields more domain-specific lemma references than random selection but a lower pass rate. Random seed selection may remain competitive because all configurations retain target-based Prover selection, so these comparisons isolate the choice of seeds rather than the value of target guidance throughout the loop.

Relation to other conjecture-selection methods. STP trains its conjecturer on conjectures that are barely provable by the current prover and pass a heuristic elegance filter, re-weighted towards the distribution of unproved statements (Dong & Ma, 2025). SGS rewards its Conjecturer with a Guide's assessment of relevance to unsolved targets and formulation quality (Bailey et al., 2026). In both methods, the selection signal is a per-conjecture proxy for training utility, used to train the Conjecturer. TACO instead keeps the Conjecturer fixed and scores whole conjecture sets by the measured target improvement of a Prover trained on them. Its only intrinsic requirement, inherited from GRPO, is that seeds be proved in some but not all of their rollouts. This direct signal costs one Prover training and evaluation per candidate set, and the utility it measures depends on the target set, performance metric, and training budget.

Limitations. Our experiments use one run per configuration and small, domain-specific evaluation sets, so comparisons between setups remain descriptive. Repeated selection on the target set may favour sampling fluctuations or target-specific gains. Held-out test improvements support transfer beyond these targets, but do not establish that each selected curriculum is reliably better than its alternatives. Each candidate curriculum requires separate Prover training and evaluation, making the procedure computationally expensive. Set-level scoring also leaves individual conjectures' contributions unresolved, while the target set determines which capabilities selection rewards. Although the iterative loop adds no human-written training problems, the initial Prover and the seed, target, and test pools depend on human-curated data.

5 Discussion and Conclusion

We introduced TACO, a target-driven framework for constructing theorem-proving curricula from formal conjectures generated directly in Lean by a separate Conjecturer. Its central methodological choice is to evaluate conjectures collectively: candidate conjecture sets are scored by the measured target-set improvement of Provers trained on them. This makes it possible to guide curriculum construction using measured training outcomes as the Prover evolves.

Our results show that model-generated formal conjectures can provide effective training curricula for neural theorem provers. Across all four mathematical domains, training on generated conjectures increases held-out test theorem coverage and overall pass rates. On the pooled subset of ten theorems solved by the initial Prover, successful proof attempts also increase, indicating improved reliability on previously solvable problems. Analysis of proof attempts also reveals more frequent references to mathematical lemmas and automation tactics already present in the initial Prover's outputs, suggesting increased use of existing proof tools.

TACO connects conjecture generation to an explicit training objective: improving Prover performance on a fixed target set. The resulting feedback guides both the choice of subsequent training problems and the Prover retained for the next iteration. The ablations show a domain-dependent benefit from target-driven seed selection. Because all configurations retain target-based Prover selection, they do not isolate the contribution of target guidance to the overall gains. Overall, our results demonstrate that curricula built from model-generated formal conjectures can improve held-out theorem proving without adding human-written training problems during the iterative loop.

AI USE STATEMENT

In this work, we used generative AI tools as coding assistants for the experimental pipeline, and to support the analysis of generated proofs and experiment logs at scale. We have not used generative AI tools to formulate or prove mathematical claims, propose hypotheses, or design the research methodology, and assistance with translation is not applicable to this work. Additionally, we used generative AI tools to create and edit software code and to help draft and edit parts of the paper. We have reviewed all AI-assisted work: all AI-assisted code was manually verified by the authors, analyses of proofs and logs were checked against the underlying data, and all text was reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or personal data. All experiments use publicly available Lean datasets and open-weight models, used in accordance with their licences. The main environmental cost is compute, which we report in Appendix E (about 940 GPU-hours). We are not aware of direct harmful applications of this work.

REPRODUCIBILITY STATEMENT

Section 3 describes the TACO loop, and Algorithm 1 (Appendix D) specifies seed selection, including rounding, ties, and capacity limits. Section 4.1 and Appendix C describe data processing and the construction of the seed, target, and test sets. Appendix E gives the model checkpoints, GRPO hyperparameters, decoding settings, Lean verification rules, Conjecturer prompts, and compute costs. The supplementary material contains the code of the TACO loop, including conjecture generation, GRPO training, target-set evaluation, and seed selection. All experiments use publicly available datasets and open-weight models.

REFERENCES

Luke Bailey, Kaiyue Wen, Kefan Dong, Tatsunori Hashimoto, and Tengyu Ma. Scaling self-play with self-guidance, 2026. URL https://arxiv.org/abs/2604.20209.

Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, Zhi-Hong Wang, Mingxuan Wang, Chenrui Wei, Shufa Wei, Huajian Xin, Fan Yang, Weihao Gao, Zheng Yuan, Tianyang Zhan, Zeyu Zheng, Tianxiang Zhou, and Thomas (Hanwen) Zhu. Seed-Prover 1.5: Mastering Undergraduate-Level Theorem Proving via Learning from Experience. *ArXiv*, abs/2512.17260, 2025. URL https://api.semanticscholar.org/CorpusID:284058256.

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zWArMedNuW.

Marco Dos Santos, Hugues de Saxcé, Haiming Wang, Mantas Baksys, Mert Unsal, Junqi Liu, Zhengying Liu, and Jia LI. Kimina Lean Server: A High-Performance Lean Server for Large-Scale Verification. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=orPt2nxLoC.

Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to Teach. In *The Sixth International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJewuJWCZ.

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, Henryk Michalewski, Nicolas Sonnerat, Demis

Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025. URL https://www.nature.com/articles/s41586-025-09833-y.

Timothe Kasriel, Thomas Lu, Devon Ding, Jingxuan He, and Dawn Song. Usefulness-Driven Learning of Formal Mathematics. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS'25*, 2025. URL https://openreview.net/forum?id=SHeRsXGRcZ.

Joshua Ong Jun Leang, Zheng Zhao, Mihaela Cătălina Stoian, Qiyuan Xu, Haonan Li, Wenda Li, Shay B. Cohen, and Eleonora Giunchiglia. Pythagoras-Prover: Advancing Efficient Formal Proving via Augmented Lean Formalisation, 2026. URL https://arxiv.org/abs/2606.12594.

Gabriel Poesia, David Broman, Nick Haber, and Noah Goodman. Learning formal mathematics from intrinsic motivation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=uNKlTQ8mBD.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

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. URL https://arxiv.org/abs/2504.11354.

Mingzhe Wang and Jia Deng. Learning to Prove Theorems by Learning to Generate Theorems. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d2a27e83d429f0dcae6b937cf440aeb1-Abstract.html.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IUikebJ1Bf0.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Zheng Yuan, Wenwei Zhang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via critic-guided search, 2025. URL https://arxiv.org/abs/2410.15700.

Huajian Xin, Daya Guo, Zhihong Shao, Z.Z. Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Advancing theorem proving in LLMs through large-scale synthetic data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=TPtXLihkny.

Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin

Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

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. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=Vcw3vzjHDb.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A TACTIC AND LEMMA USAGE

Table 3 reports the percentage of held-out test proof attempts containing selected tactics or explicit lemma references. Each evaluation contains 1024 attempts (32 problems, 32 attempts per problem), including successful and unsuccessful attempts. Each name is counted at most once per attempt; comments and strings are excluded, and qualified and unqualified lemma names are grouped. These counts measure explicit mentions, not successful execution.

Each cell reports Base → Iter. 10. Since all setups start from the same Prover, a shared Base evaluation is used within each domain. For each domain, we select the three most frequent automation tactics and the three most frequent domain-relevant lemma names in the final targeted/score Prover, then report the same names across all setups.

Here, *targeted* denotes target-aligned initial seeds and *random* denotes the random initial seed set. The second component denotes target-driven (*score*) or random iterative seed selection. All setups retain target-based best-Prover selection.

Table 3: Explicit tactic and lemma mentions across domains and setups. Each cell reports Base → Iter. 10 as percentages of 1024 held-out test proof attempts, including unsuccessful attempts. A shared Base evaluation is used within each domain. The three tactics and three domain-relevant lemma names most frequent in the final targeted/score Prover are reported across all setups. Each name is counted once per attempt, excluding comments and strings; qualified and unqualified lemma names are grouped. Mentions do not imply successful execution. Geometry lemmas concern auxiliary arithmetic facts.

	Automation tactics			**Lemma references**		
Radical algebra						
Setup	`nlinarith`	`norm_num`	`positivity`	`sq_sqrt`	`sqrt_nonneg`	`sqrt_pos`
targeted/score	95.4 → 99.9	36.3 → 51.3	26.2 → 27.7	65.0 → 97.7	14.3 → 79.6	1.8 → 2.1
targeted/random	95.4 → 96.7	36.3 → 55.6	26.2 → 9.9	65.0 → 92.5	14.3 → 51.4	1.8 → 6.9
random/score	95.4 → 99.7	36.3 → 45.4	26.2 → 17.7	65.0 → 80.3	14.3 → 31.2	1.8 → 8.2
random/random	95.4 → 98.1	36.3 → 41.8	26.2 → 23.2	65.0 → 72.1	14.3 → 34.0	1.8 → 9.2
Geometry of areas						
Setup	`norm_num`	`ring_nf`	`field_simp`	`pi_pos`	`sqrt_eq_iff_sq_eq`	`sqrt_nonneg`
targeted/score	57.9 → 96.6	15.0 → 88.9	6.9 → 85.1	1.6 → 16.3	0.6 → 11.9	1.6 → 6.1
targeted/random	57.9 → 65.6	15.0 → 19.8	6.9 → 91.6	1.6 → 42.9	0.6 → 17.0	1.6 → 25.4
random/score	57.9 → 97.9	15.0 → 79.4	6.9 → 27.5	1.6 → 38.5	0.6 → 6.2	1.6 → 21.7
random/random	57.9 → 54.5	15.0 → 90.1	6.9 → 76.9	1.6 → 35.2	0.6 → 7.5	1.6 → 5.7
Geometry of volumes						
Setup	`norm_num`	`linarith`	`simp`	`pi_pos`	`pi_ne_zero`	`floor_eq_iff`
targeted/score	56.5 → 99.9	18.8 → 96.3	13.9 → 30.3	0.4 → 48.8	0.3 → 33.3	0.4 → 15.2
targeted/random	56.5 → 91.7	18.8 → 71.3	13.9 → 0.1	0.4 → 29.2	0.3 → 6.0	0.4 → 2.1
random/score	56.5 → 75.8	18.8 → 54.2	13.9 → 13.3	0.4 → 24.5	0.3 → 0.5	0.4 → 6.1
random/random	56.5 → 79.4	18.8 → 47.2	13.9 → 2.6	0.4 → 26.9	0.3 → 0.5	0.4 → 10.0
Trigonometry						
Setup	`nlinarith`	`norm_num`	`simp`	`cos_sq`	`sin_sq`	`sin_sq_add_cos_sq`
targeted/score	21.6 → 56.5	39.9 → 42.5	27.4 → 31.2	3.4 → 33.2	3.8 → 32.7	4.3 → 20.6
targeted/random	21.6 → 48.8	39.9 → 59.2	27.4 → 24.9	3.4 → 20.1	3.8 → 20.6	4.3 → 16.3
random/score	21.6 → 81.6	39.9 → 15.3	27.4 → 15.9	3.4 → 3.2	3.8 → 3.1	4.3 → 8.9
random/random	21.6 → 21.1	39.9 → 58.2	27.4 → 33.0	3.4 → 5.9	3.8 → 6.1	4.3 → 8.3

B EXAMPLES OF SEEDS, CONJECTURES, AND TARGETS

Table 4 and 5 illustrate curriculum content for the targeted/score setup. All variables are real. Each table presents an initial seed, conjectures generated at iterations 1, 5, and 10, and a target problem.

All displayed conjectures have archived proofs from the experiments. They are selected from candidate curricula, not necessarily the curriculum that produced the retained Prover, and do not form a single parent–child lineage.

Table 4: Radical-algebra examples and qualitative interpretation. The conjectures exercise bounds, products, and compositions of square roots; the target couples three variables through a shared constraint.

Stage	Statement	Qualitative interpretation
Initial seed	For $0 \le x \le 4$, $\sqrt{x} \ge x/2$.	An elementary bound obtained from nonnegativity and the square-root square identity. The seed-selection evaluation recorded 12/16 successes.
Conjecture, iter. 1	For $0 \le x \le 4$, $\sqrt{x} \ge x/2 \ \wedge \ \sqrt{x} \le 2$.	Adds an upper bound to the seed. The archived proof applies `nlinarith` to both goals using square-root facts.
Conjecture, iter. 5	For $0 < x < 1$, $\sqrt{x}(1-\sqrt{x}) \le 1/4$.	Bounds a product by completing the square in $\sqrt{x}$. The archived proof supplies $(1/2 - \sqrt{x})^2 \ge 0$ to `nlinarith`.
Conjecture, iter. 10	For $0 < x \le 1$, $\dfrac{\sqrt{x}+x}{2} \ge \sqrt{x\sqrt{x}}$.	Introduces a nested radical and its nonnegativity obligations. It remains an instance of AM–GM, so greater structural complexity does not necessarily imply greater mathematical difficulty.
Target	For $a, b, c > 0$ and $a+b+c=1$, $\sqrt{a+bc} + \sqrt{b+ca} + \sqrt{c+ab} \le 2$.	Couples three radical expressions through a normalisation constraint. Successful target proof attempts increase from 0/32 at Base to 17/32 at iteration ten.

Table 5: Trigonometric examples and qualitative interpretation. Standard identities support algebraic transformations and inequalities.

Stage	Statement	Qualitative interpretation
Initial seed	$\cos(2x) = 2\cos^2 x - 1$.	A standard double-angle identity, directly expressible using a Mathlib lemma.
Conjecture, iter. 1	$\cos(2x) = 2\cos^2 x - 1$ $\wedge \ \cos(2x) = 1 - 2\sin^2 x$.	Combines two equivalent forms of the double-angle identity. The archived proof uses trigonometric rewriting and polynomial normalisation.
Conjecture, iter. 5	$\cos(2x)\cos x = 2\cos^3 x - \cos x$.	Multiplies the seed identity by $\cos x$. The archived proof is a double-angle rewrite followed by `ring`: an extra algebraic step, not a new identity to learn.
Conjecture, iter. 10	For $0 \le B \le \pi/2$, $\sin^2 B + \cos^2 B + 2\sin B \cos B \le 2$.	Combines $\sin^2 B + \cos^2 B = 1$ with a nonnegative square and `nlinarith`. The stated interval assumption is unnecessary for this inequality.
Target	$\cos(a+b)\sin b - \cos(a+c)\sin c$ $= \sin(a+b)\cos b - \sin(a+c)\cos c$.	Requires coordinating addition formulas and algebraic cancellation. The archived final proof uses trigonometric simplification and `ring`. Successes increase from 1/32 to 23/32.

C Domain Selection and Evaluation-Set Construction

We constructed domain-specific problem pools from our evaluation pool, comprising problems from publicly available Lean datasets: NuminaMath-LEAN, Lean-Workbook-Plus, and Pythagoras-LM. Deduplication by exact statement text reduced 126,429 entries to 26,418 distinct statements. Here,

evaluation pool refers to this full source collection; the smaller, domain-specific target and test sets used in the experiments are constructed from it as described below.

Domain membership was determined from the statements, without using reference proofs:

- **Radical algebra:** real scalar statements containing square roots (`Real.sqrt`, $\sqrt{\ }$, or unqualified `sqrt`), with structural exclusions to remove trigonometric and logarithmic/exponential expressions, π, fractional-power syntax, and specified non-scalar structures.
- **Geometry of areas:** the token `area` in an identifier introduced by `let`.
- **Geometry of volumes:** the token `volume` in an identifier introduced by `let`.
- **Trigonometry:** a statement containing `Real.sin`, `Real.cos`, `Real.tan`, `Real.arcsin`, `Real.arccos`, or `Real.arctan`.

C.1 Difficulty and Automation Filtering

Our objective was to construct two candidate pools within each domain: easy problems to initialise the seeds, and harder problems to form the target and test subsets. We used existing evaluations in which the Base Prover generated 16 proof attempts per statement. The number of successful attempts provided an empirical difficulty score.

- **Seed candidates:** a score of at least 6/16, indicating problems already reliably solved by the Base Prover.
- **Target/test candidates:** a score of at most 1/16, indicating problems rarely or never solved in these initial evaluations.
- Problems scoring 2/16 to 5/16 were initially assigned to neither pool.

Figure 3 shows the distribution of these scores across the **entire test set of 26,418 distinct statements**, before domain filtering or automation filtering. Each step gives the percentage of statements with exactly that score; it is not a cumulative distribution.

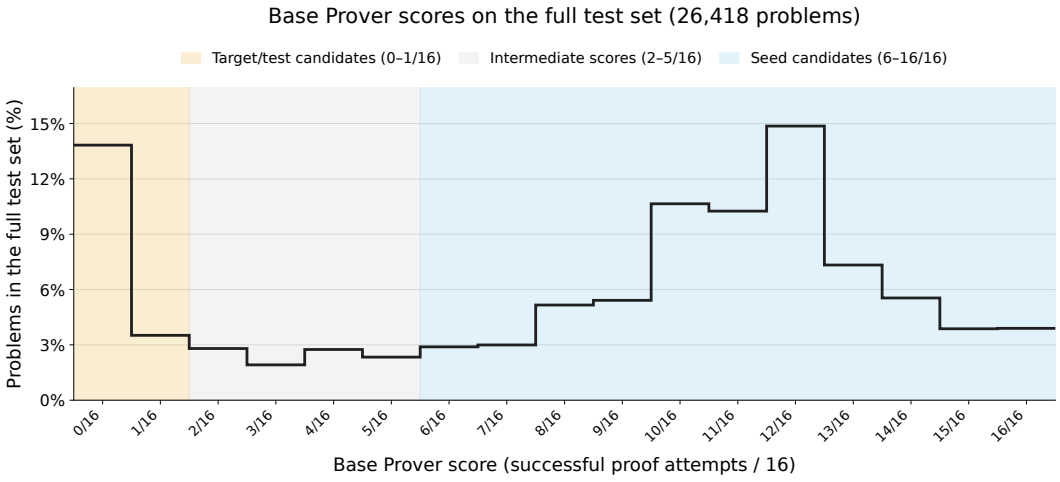


Figure 3: Distribution of Base Prover scores across the full test set of 26,418 distinct statements. Initial evaluation scores range from 0/16 to 16/16. The shaded ranges identify candidates for target/test subsets (0–1) and seeds (6–16); domain membership and subsequent selection further restrict these pools.

Across the full test set, 4,582 statements (17.34%) score 0–1/16, 2,589 (9.80%) score 2–5/16, and 19,247 (72.86%) score 6–16/16. These counts precede all domain and automation filters.

We applied an additional automation filter only to the target/test candidates, not to seed candidates. Its purpose was to remove problems that received a low Base Prover score but could nevertheless be solved by a predefined Lean script. Examples of scripts tested are:

```
norm_num <;> ring_nf <;> field_simp <;> ring_nf

intros <;> simp_all <;> norm_num at * <;> nlinarith
```

All four domains used the same bank of 16 scripts. Each script had a budget of 100,000 heartbeats and 30 seconds in Lean 4.15.0 with Mathlib and Aesop. A candidate was excluded as soon as one script produced a complete accepted proof.

Table 6: Domain pools and initial target/test candidate filtering. Candidates have a Base Prover score of at most 1/16; exclusions correspond to complete proofs found by the fixed automation bank. Counts precede target/test partitioning and the area-domain supplement described in the text.

Domain	Domain pool	Candidates $\leq 1/16$	Excluded by automation	Remaining
Radical algebra	247	160	6	154
Geometry of areas	660	124	61	63
Geometry of volumes	543	98	33	65
Trigonometry	244	188	48	140

C.2 Final Seeds, Targets, and Tests

Each domain contains 16 target-aligned initial seeds, 32 target problems, and 32 test problems. When more than 64 target/test candidates remained, selection prioritised low initial scores and similarity between statements. Target and test sets were constructed jointly by pairing similar statements within difficulty and audit-status strata, then balancing statement features and lengths between the two splits. Similarity used statement features only; reference proofs did not determine selection or partitioning.

In all four domains, the 16 initial seeds are sampled uniformly without replacement from the seed candidates.

D Seed selection algorithm

Algorithm 1 specifies step 4 of Section 3. Each set $\widehat{\mathcal{C}}_j^{(i)}$ contains the eligible conjectures of candidate set j, i.e. those proved in some but not all of their rollouts during training of the corresponding Prover variant. Sets scoring above the mean receive seeds in proportion to their excess over the mean; if all scores tie, seeds are allocated uniformly. Proportional shares are rounded by the largest-remainder method: each set first receives the integer part of its share, and the remaining seeds go to the sets with the largest fractional parts, so that exactly B seeds are allocated. Each allocation is then capped by the number of eligible conjectures in the set, and any shortfall is filled from the remaining sets in decreasing order of score. A set with no eligible conjecture therefore receives no seeds, and the next seed set contains fewer than B seeds only if fewer than B conjectures are eligible in total. In the random-selection ablation (Section 4.2), the procedure is replaced by sampling B conjectures uniformly without replacement from $\bigcup_j \widehat{\mathcal{C}}_j^{(i)}$.

E Training, generation, and compute details

Models. The Prover is initialised from the base Qwen3.5-2B checkpoint and fine-tuned as described in Section 4.1. The Conjecturer is Qwen3.8-27B in FP8, served with vLLM and kept fixed throughout.

Conjecture generation. The Conjecturer receives one of three system prompts (close, intermediate, exploratory; Figure 4) and the seed statement as user message. Each call requests eight statements. For each seed, calls are repeated until 64 distinct conjectures are collected (32 close, 16 intermediate and 16 exploratory), where distinctness is exact string match. We allow at most 64 calls per seed. If this limit is reached first, the candidate set contains fewer than 64 conjectures and

Algorithm 1 Seed selection at iteration i.

Require: scores $s_1, \dots, s_n$; eligible sets $\widehat{\mathcal{C}}_1^{(i)}, \dots, \widehat{\mathcal{C}}_n^{(i)}$; budget B

$\bar{s} \leftarrow \frac{1}{n} \sum_j s_j$
$w_j \leftarrow \max(s_j - \bar{s},\, 0)$ for all j
if $w_j = 0$ for all j **then** ▷ all scores tie
    $w_j \leftarrow 1$ for all j
end if
$a_j \leftarrow$ largest-remainder rounding of $B\, w_j / \sum_l w_l$ ▷ $\sum_j a_j = B$
$a_j \leftarrow \min\left(a_j,\, |\widehat{\mathcal{C}}_j^{(i)}|\right)$ for all j
for j in decreasing order of s_j **do** ▷ fill any shortfall
    $a_j \leftarrow a_j + \min\left(|\widehat{\mathcal{C}}_j^{(i)}| - a_j,\; B - \sum_l a_l\right)$
end for
Sample a_j conjectures uniformly without replacement from each $\widehat{\mathcal{C}}_j^{(i)}$
return their union as $\mathcal{S}^{(i+1)}$

is used for training as is. Sampling uses temperature 0.8, top-k 20, top-p 1.0 and presence penalty 2.0.

GRPO training. At each iteration, a copy of the current base Prover is trained on each candidate set for one epoch, so that each conjecture is used once as a prompt, with the hyperparameters of Table 7.

Reward and verification. Each proof is checked by the Kimina Lean Server (Lean 4.15.0, Mathlib) with a 120-second timeout. A proof attempt receives reward 1 if it is accepted without `sorry`, and 0 otherwise.

Compute. All experiments run on a single NVIDIA GH200 GPU. A complete iteration, comprising conjecture generation, up to 16 GRPO runs, their evaluation on the target set, and seed and Prover selection, takes about six hours, so a ten-iteration run takes about 60 GPU-hours and the 16 reported runs (four domains, four configurations) about 940 GPU-hours.

Table 7: GRPO hyperparameters, identical across candidates, iterations, and domains.

Epochs per candidate set	1
Rollouts per conjecture (group size)	8
Per-device batch size / gradient accumulation	2 / 1
Learning rate (constant)	10^{-7}
KL coefficient β	0
Maximum completion length	2,048 tokens
Sampling	temperature 1.0, top-k 20, top-p 1.0, presence penalty 2.0
Precision	bf16, gradient checkpointing

```
Write exactly eight conjectures in Lean 4 with Mathlib.

CLOSE: Keep the seed's structure, making small changes to its numerical data,
assumptions, or construction. Prefer a small extra reasoning step over only
changing constants, while staying close to the seed's difficulty.

Every conjecture must stay in the specific mathematical domain of the user's seed.
Make them independent, explicitly type and declare every variable, and avoid
equivalent rewrites, tautologies, or assuming the conclusion.
You may mix ∀/→ statements with theorem parameters and named hypotheses (h₀ : ...).

Assume import Mathlib, import Aesop, and open BigOperators Real Nat Topology Rat.
Return only eight declarations named conjecture, each ending in := by, separated
by a blank line. No proofs, sorry, imports, markdown, or commentary.

This example illustrates proximity and format. Follow the actual seed's domain;
do not switch to number theory unless that is its domain.

EXAMPLE INPUT
theorem statement : ∀ n : ℤ, 3 ∣ n → 3 ∣ n + 6 := by

EXAMPLE OUTPUT
theorem conjecture : ∀ n : ℤ, 3 ∣ n → 3 ∣ n + 9 := by
theorem conjecture (n : ℤ) (h₀ : 4 ∣ n) : 4 ∣ n + 8 := by
[... six more ...]
END OF EXAMPLE

Now generate eight CLOSE conjectures for the user's seed.
```

Figure 4: Conjecturer system prompt (CLOSE). The INTERMEDIATE and EXPLORATORY prompts replace the second paragraph and the example outputs, as follows. **Intermediate:** "Change the task while keeping a clear connection to the seed. Make it slightly harder by adding one useful reasoning step, such as combining the seed's idea with another elementary fact from the same domain." **Exploratory:** "Explore other problem families within the seed's specific domain, without keeping its structure. Aim slightly above the seed's difficulty, for example by combining two elementary ideas, without a large jump in difficulty."