MPS-Prover: Advancing Stepwise Theorem Proving by Multi-Perspective Search and Data Curation

Zhenwen Liang[1]*, **Linfeng Song**[1], **Yang Li**[2], **Tao Yang**[2], **Haitao Mi**[1], **Dong Yu**[1]

[1]Tencent AI Lab [2]Tencent LLM Department

Abstract

Automated Theorem Proving (ATP) in formal languages remains a formidable challenge in AI, demanding rigorous logical deduction and navigating vast search spaces. While large language models (LLMs) have shown promising performance, existing stepwise provers often suffer from biased search guidance, leading to inefficiencies and suboptimal proof strategies. This paper introduces the Multi-Perspective Search Prover (MPS-Prover), a novel stepwise ATP system designed to overcome these limitations. MPS-Prover incorporates two key innovations: a highly effective post-training data curation strategy that prunes approximately 40% of redundant training data without sacrificing performance, and a multi-perspective tree search mechanism. This search integrates a learned critic model with strategically designed heuristic rules to diversify tactic selection, prevent getting trapped in unproductive states, and enhance search robustness. Extensive evaluations demonstrate that MPS-Prover achieves state-of-the-art performance on multiple challenging benchmarks, including miniF2F and ProofNet, outperforming prior 7B parameter models. Furthermore, our analyses reveal that MPS-Prover generates significantly shorter and more diverse proofs compared to existing stepwise and whole-proof methods, highlighting its efficiency and efficacy. Our work advances the capabilities of LLM-based formal reasoning and offers a robust framework and a comprehensive analysis for developing more powerful theorem provers.

1 Introduction

Automated Theorem Proving (ATP) is the task of automatically generating formal proofs for given mathematical or logical statements. By transforming problems into theorems in a formal language (e.g., Lean [Moura and Ullrich, 2021] or Isabelle [Paulson, 1994]) and recursively interacting with the proof assistant's engine to construct full proofs, an ATP system generates machine-verified proofs that guarantee strict logical correctness. This verifiability makes ATP indispensable for formal verification of solutions and proofs, where each reasoning step must be checked rigorously. ATP has long been viewed as a foundational and challenging problem in both AI and mathematics, as such systems can leverage massive computational power, potentially aiding mathematicians in evaluating new hypotheses and even solving open mathematical problems. The rapid progress of large language models (LLMs) has significantly advanced automated theorem proving (ATP), exemplified by AlphaProof's silver medal performance at IMO 2024 [AlphaProof and Teams, 2024].

Recent research tackles these challenges by combining the reasoning abilities of LLMs with feedback from proof checkers (e.g., the Lean compiler). Two main approaches have emerged. One is *whole-proof generation* [Wang et al., 2024c, Xin et al., 2024, Lin et al., 2025, Zhang et al., 2025, Wang et al., 2025], where the LLM attempts to output an entire proof script in one shot. This leverages the model's ability to plan with a high-level view but forgoes intermediate verification, making it

*Contact: zhenwzliang@global.tencent.com

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

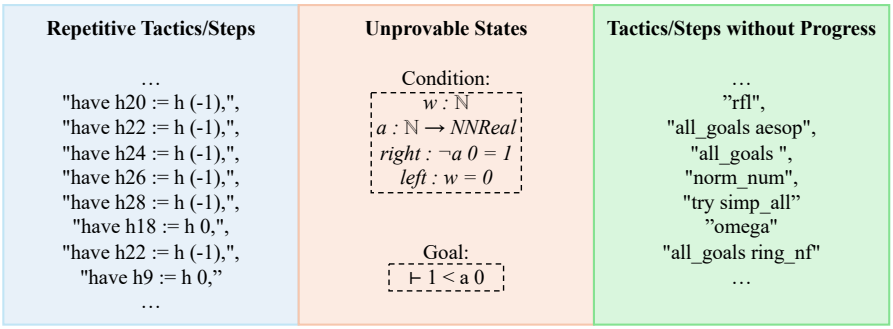


Figure 1: Common failure patterns in step-based theorem provers. (a) Repetitive steps caused by critique model over-preference for specific tactics. (b) Unprovable states resulting from incorrect tactic choices that overly simplify conditions. (c) Ineffective tactic applications that fail to make progress.

prone to failures on long or intricate proofs. The second, and the focus of this paper, is *stepwise formal proof generation* [Wu et al., 2024, Li et al., 2024b, Xin et al., 2025]. Here, an LLM iteratively proposes the next proof step (a formal tactic) given the current proof state. After each step, a formal proof assistant verifies the result, ensuring the proof stays on a correct path and providing an updated proof state as feedback. This step-by-step strategy offers several advantages: it allows for continuous interaction with the proof engine, enables progressive simplification of the search target, offers higher fault tolerance (errors only require backtracking to the previous step, not restarting the entire proof), and naturally lends itself to tree search methods that explore different proof paths.

However, stepwise LLM-based provers face their own key challenges, as illustrated in Figure 1. First, the critic model guiding node selection in tree search can become biased. This bias often stems from the high frequency of certain "safe" or broadly applicable tactics (e.g., $have$, or general-purpose simplifiers like $aesop$ and $simp\_all$ when part of a successful sequence) in the training data. These tactics, while often part of valid proofs and less prone to immediate errors, may not always lead to the most efficient or even a correct overall proof path if the model over-relies on them, leading to stalled progress from similar tactic suggestions (Figure 1a). Second, incorrect tactic applications can lead to unprovable states by oversimplifying conditions (Figure 1b). Third, powerful but conditionally effective tactics (e.g., $aesop$, $simp\_all$) might be applied ineffectively. LLMs may propose these due to their frequent appearance in the training data or their ability to produce local simplifications that seem promising, yet they can make no progress or even obscure the path forward when the state is not genuinely suitable for such simplification (Figure 1c). While Best-First Search (BFS)-based methods [Li et al., 2024b, Xin et al., 2025] have shown promise in navigating this search space, their typical reliance on a single critic score for node expansion can still render them vulnerable to these failure modes, particularly the biases inherent in learned critics.

In this paper, we introduce the *Multi-Perspective Search Prover (MPS-Prover)*, a novel approach designed to overcome these limitations and significantly enhance stepwise proving performance. Our first contribution is a carefully designed post-training data curation strategy. Unlike existing expert iteration approaches that uniformly add all newly proved problems, we introduce explicit rules to filter out approximately 40% of redundant or low-value training examples, focusing the model on learning more complex reasoning patterns. This curates a higher-quality training set, leading to improved model accuracy and mitigating overfitting, especially when augmented with natural language reasoning datasets. Our second core contribution, building upon Best-First Search (BFS) methodologies [Li et al., 2024b, Xin et al., 2025], is a multi-perspective tree search enhanced with strategically devised heuristic critiques. These critiques diversify tactic selection, reducing the risk of becoming trapped in repetitive, unproductive, or unprovable states by encouraging broader exploration during proof search.

Our experiments demonstrate that MPS-Prover achieves state-of-the-art performance across multiple ATP benchmarks, including miniF2F and the more challenging ProofNet. On miniF2F, MPS-Prover surpasses previous stepwise provers. Furthermore, on ProofNet, within the 7B model class, MPS-Prover outperforms all baselines, including those employing CoT reasoning. Our analyses also reveal that MPS-Prover generates significantly shorter and more diverse proofs compared to both

BFS-based stepwise provers (under equivalent computational budgets) and leading whole-proof provers, highlighting the efficiency and efficacy of our multi-perspective search strategy. More specifically, the average solution length produced by MPS-Prover is only 3.44, compared to 15.91 and 52.16 for Kimina-Prover and Deepseek-Prover V2, respectively. These findings illustrate key advantages of our enhanced stepwise approach and suggest promising directions for future hybrid prover development. The primary contributions of this paper are:

1. A novel post-training data curation strategy for stepwise provers, effectively eliminating approximately 40% of redundant training data while achieving superior performance.

2. The Multi-Perspective Search Prover (MPS-Prover), an innovative tree search method with heuristic critiques to enhance tactic diversity, prevent critique model bias, and improve search robustness.

3. Demonstration of state-of-the-art performance by MPS-Prover on multiple benchmarks, including miniF2F and ProofNet, along with analyses showing generation of shorter and more diverse proofs.

2 Method

2.1 Expert Iteration on Tactic Generation

Most recent Automated Theorem Proving (ATP) systems utilize an expert iteration [Polu and Sutskever, 2020] process to collect training data, which consists of several key steps: (1) autoformalization of natural language problems into formal proofs; (2) attempting proofs or disproofs using the model trained in the previous iteration; and (3) integrating newly proved theorems and the proofing steps into subsequent training iterations.

We follow previous work to collected public available natural language problems and formal theorems and their proofs, including the Lean Workbook [Ying et al., 2024], Numina [Li et al., 2024a] and AoPS-Instruct [Mahdavi et al., 2025] for expert iteration. After formalizing the natural language problems, we conducted 26 iterative rounds of expert iteration. This process yielded over 30,000 proven theorems and approximately 6 million individual proving (state, step) pairs that can be used to train our prover.

2.2 Training Data Curation

Filtering Short Proofs. To focus the model's learning on these more complex reasoning patterns and reduce its reliance on simple tactics, we exclude theorems from the training set that can be proven in 3 steps or fewer. The number 3 is determined by a grid search on {2, 3, 4, 5}.Our analysis indicated that these very short proofs predominantly rely on a limited set of elementary tactics (e.g., rfl, $simp\_all$, or $nlinarith$) and thus offer minimal insight into advanced problem-solving techniques. By removing these overly simplistic examples, we reduced our initial training dataset by approximately 40%. It is important to note that filtering these simple proofs is not expected to degrade the model's ability to solve easy problems. This is because the training data for a step-wise prover inherently includes a vast number of "late-stage" proof steps. These steps, taken when a proof is already well underway and nearing completion, often resemble the states encountered in simpler problems. Consequently, the model still receives ample exposure to simpler reasoning contexts through these intermediate steps of complex proofs.

Removing Ineffective Tactics. We additionally filter out the training data where the step does not meaningfully advance the proof state. Certain tactics intended for simplification occasionally do not bring any change to the proof state, such as $aesop$, all_goals, and $simp\_all$. After evaluating our dataset, we identified and removed about 5% of such ineffective tactics. This targeted pruning encourages the model to better discern when these simplification tactics should be applied, reducing overreliance and improving proof efficiency.

2.3 Multi-Perspective Tree Search

As illustrated in Figure 2, the traditional BFS approach selects nodes based solely on the best critic scores. Following Li et al. [2024b], our critic model is trained using a hierarchical, tree-based distance prediction method to enhance its guidance capabilities during proof searches. The output from the

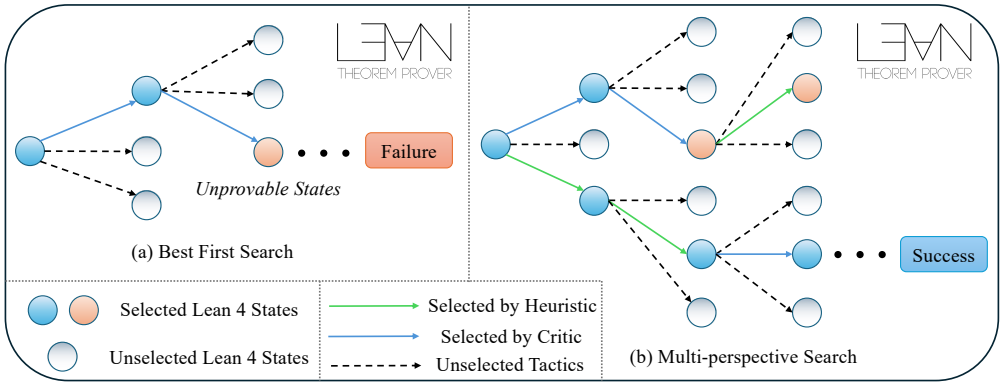


Figure 2: Search-strategy comparison in Lean-based proving. (a) Best-First Search follows the single branch favoured by a learned critic; when that critic's inherent bias selects an unprovable state or an ineffective tactic, the entire proof attempt terminates in failure. (b) Multi-Perspective Search (MPS) evaluates each expansion step from heuristics as well as the critic, preserving a more diverse set of promising Lean 4 states and steering the prover around dead ends and toward a successful proof.

critic model guides the proof search tree by indicating proximity to the completion of the proof—the smaller the predicted distance, the closer the node is to a successful proof.

While the critic model significantly enhances decision-making during most search steps, it can sometimes fail, resulting in unprovable states and reduced search efficiency. For instance, the proof-by-contradiction tactic, although powerful, can substantially alter the proof goal and make the theorem unprovable if misapplied. Additionally, sometimes critic models tend to frequently propose similar tactics, creating repetitive and ineffective local minima. As illustrated in Figure 2(a), the proof search is guided by the critic model to reach an intermediate node where the state becomes unprovable, leading to wasted effort on subsequent steps and ultimately resulting in a failed proof attempt. To enhance the diversity of the guiding signals, we introduce three heuristic selection rules:

Tactic Effectiveness Scoring. We assign different scores to tactics based on their perceived efficacy in advancing the proof. Generally, tactics that introduce new, substantive assumptions or significantly restructure the proof goal, such as $rcases$, $intro$, $contrapose$, $induction$, or proof by $contradiction$ (when appropriately applied), are assigned higher scores. These are often tactics that can unlock new reasoning pathways or simplify the problem by breaking it down. Conversely, auxiliary tactics or those focused on more localized simplifications, like $norm\_num$ and $simp\_all$, receive lower scores. While tactics like proof by contradiction can be problematic if misapplied by the critic model alone (as noted earlier), its inclusion with a high score in this heuristic perspective ensures it remains a viable option for exploration when potentially beneficial. These scores are manually assigned based on human expert experience in Lean theorem proving. A detailed table of these tactic scores is provided in Section 2.4.

Minimizing Case Splits. We select tactics that result in the fewest occurrences of `case` within the state string. While tactics like `induction`, `constructor`, and `split` are beneficial under specific circumstances, excessive case splitting complicates proof states. This heuristic encourages simpler, more manageable proof states.

Shortest State Preference. We prioritize tactics leading to shorter Lean 4 state strings. Similar to minimizing cases, this heuristic favors simpler, more straightforward states, facilitating more efficient proof completion.

As shown in Figure 2, our tree search maintains up to four nodes for each expansion step. Specifically, from the set of nodes selected in the previous iteration, we generate $N_{samples}$ candidate tactics for each using the LLM. This results in a larger pool of potential next states (e.g., if 4 nodes were selected and $N_{samples} = 8$, we'd have $4 \times 8 = 32$ candidate next states). From this expanded pool: 1. One node is selected based on the critic model's prediction (i.e., the one with the smallest predicted distance to completion). 2. Three additional nodes are selected based on our heuristic rules. Each heuristic rule evaluates all candidate next states and picks the one that best satisfies its criterion. If

different perspectives select the same node, we only retain it once, meaning that in such cases, fewer than four unique nodes might be carried forward to the next search iteration.

2.4 Tactic Effectiveness Scoring

The Tactic Effectiveness Scoring heuristic in MPS-Prover assigns a numerical score to potential next steps based on the tactic used. These scores are designed to prioritize tactics that are generally more impactful or transformative in the proof process, while giving lower scores to auxiliary or very general-purpose tactics that might be applied speculatively. The scores are based on common patterns observed in mathematical proofs and expert experience with the Lean theorem prover. The goal is to guide the search towards more direct and structured proofs by favoring steps that represent significant logical advancements.

Below is the scoring table used. Tactics are grouped by score, with higher scores indicating a stronger preference. Note that regular expressions are used for some tactic patterns (e.g., $simp??$ matches $simp$ and $simp?$).

Table 1: Tactic Effectiveness Scores

Score	Tactics / Patterns
6	`exact`, `refine`, `rintro`, `rcases`, `induction`, `revert`, `by_contra`, `contrapose`
5	`rw`, `rw .* at`, `convert`, `apply`, `subst`, `linarith`, `congr`, `ring_nf`
4	`ring`, `field_simp`, `group`, `aesop`
3	`simp??`, `simp_all`, `simp only`
2	`norm_cast`, `push_cast`, `clear`
1	`norm_num`, `swap`, `all_goals`
0	`have x = y` (without a subsequent *by* block for proof)

Score 6 (Highly Transformative/Goal-Closing): This tier includes tactics that often conclude a proof branch directly (e.g., $exact$, $refine$), introduce crucial case distinctions or structural changes (e.g., $induction$, $rcases$), or fundamentally alter the goal's form (e.g., by_contra, $contrapose$, $revert$). These are typically strong indicators of significant progress.

Score 5 (Strong Rewriting/Application): Tactics like rw (rewrite), $apply$ (apply a hypothesis/lemma), and $convert$ (change goal to a definitionally equal one) are powerful for making targeted changes. Algebraic simplification tools like $linarith$ and $ring\_nf$ also fall here as they can often solve subgoals involving arithmetic or ring structures.

Score 4 (Domain-Specific Solvers/Automated Tactics): This includes more specialized solvers like $ring$ (for ring equalities) and $field\_simp$ (for field simplifications), $group$ (for group theory), as well as general automated tactics like $aesop$. While powerful, $aesop$ is placed slightly lower than the top tier as it can sometimes be a "black box" and its success is highly conditional.

Score 3 (General Simplification): The $simp$ family of tactics ($simp$, $simp\_all$, $simponly$) are general-purpose simplification tools. They are very useful but are scored moderately because they can sometimes be applied excessively or ineffectively, leading to many unproductive steps. For tactics that are excluded in the table, we assign a default score of 3.

Score 2 (Normalization/Cleanup): Tactics like $norm\_cast$ (normalize casts), $push\_cast$ (push casts inwards/outwards), and $clear$ (remove unused hypotheses) are important for maintaining a clean and manageable proof state but don't usually represent major logical steps forward.

Score 1 (Auxiliary/Low Impact): This includes very basic numerical normalization ($norm_num$), reordering goals ($swap$), or meta-level tactic combinators (all_goals) which are generally supportive rather than primary drivers of proof progress.

Score 0 (Potentially Redundant $have$): A $havex = y$ statement without an accompanying by block to prove it (implying it might be proved by a trivial step, or is simply a renaming) is given the lowest score. The intent here is to penalize the simple act of stating a trivial equality without further justification as a standalone step.

This heuristic scoring aims to complement the learned critic model by providing a stable, experience-based bias towards tactics that are historically effective in structuring and advancing mathematical proofs. We acknowledge that each heuristic rule, including the critic model, has inherent biases and limitations, favoring certain proof tactics or states. However, by concurrently applying these criteria, we substantially enhance the diversity of each search layer, ensuring promising nodes are retained rather than overlooked due to single-criterion bias.

3 Experiment

Benchmarks To comprehensively evaluate our prover, we utilize two widely recognized benchmarks. 1. miniF2F [Zheng et al., 2022]: This is the standard benchmark in the ATP community. The problems are sourced from mathematics competitions (AMC, AIME, IMO) as well as high-school and undergraduate curricula. We use the latest version available from the Huggingface Numina repository[2], which corrects eight errors identified in the original dataset. 2. ProofNet [Azerbayev et al., 2023]: This benchmark consists of 371 problems, characteristic of undergraduate-level mathematics. We report performance on its test split.

Supervised Fine-tuning We employ supervised fine-tuning (SFT) on Qwen2.5-Math-7B-base. The SFT dataset is a composition of: (1) step-by-step proof data generated during expert iteration and curated by our proposed filtering techniques; (2) whole proof data, formed by concatenating the accepted proof steps; and (3) data for training the distance critic model for our search algorithm. This aggregated dataset amounts to approximately 3.5 million question-answer pairs after applying our training data filtering method. The model was trained for 3 epochs using a cosine learning rate scheduler with a maximum learning rate of 2×10^{-5}. We utilized a cumulative batch size of 256. The training was performed on 8 * H20 80G GPUs, with a total training duration of about 3 days.

Evaluation Setup All evaluations are conducted using Lean version 4.16.0. For interaction between the LLM and the Lean proof assistant, we utilize the *repl* tool[3], which facilitates step-by-step execution. To ensure the rigorous correctness of generated proofs, especially since repl might occasionally misinterpret certain erroneous steps as valid during step-wise execution, we perform a final verification. This involves concatenating all generated steps to form a complete proof script, which is then checked by the Lean compiler. The timeout for executing a whole search is set to 3600 seconds, while the per-step tactic execution timeout is 60 seconds. Our search budget per problem is defined by the total number of tactic candidates explored. This is given by $N_{pass} \times N_{perspectives} \times N_{max\_iter} \times N_{samples}$, where N_{pass} is the number of independent search trials for a problem (e.g., for pass@k, $N_{pass} = k$), $N_{perspectives} = 4$, $N_{max\_iter} = 800$ (maximum search iteration), and $N_{samples} = 8$ (LLM samples per selected node). The "accumulative" search strategy, following Xin et al. [2025] (BFS-Prover), denotes an incremental evaluation protocol to assess the model's maximum potential. In this setup, we keep searching and each search iteration focuses exclusively on problems that were not solved in prior iterations.

3.1 Main Results

We conduct extensive experiments to evaluate MPS-Prover against state-of-the-art methods on standard benchmarks. Our primary results on miniF2F are summarized in Table 2, with baseline details in Appendix B. Our method achieves the best performance among all step-level solvers evaluated. Specifically on miniF2F, MPS-Prover successfully proves 185 out of 244 problems (75.82% accuracy), demonstrating a significant improvement over the previous state-of-the-art step-prover, BFS-prover.

When considering all models within the 7B parameter class (both whole-proof and step-wise), our model's performance is only surpassed by DeepSeek-Prover-V2 (Distilled, CoT). We posit this is expected, as their 7B model is distilled from a significantly larger model, a process known to

[2] `https://huggingface.co/datasets/AI-MO/miniF2F_test`

[3] `https://github.com/leanprover-community/repl`

Table 2: Comparison of Whole-Proof and Step-level Provers on miniF2F-test.

Method	Model Size	Sample Budget	Accuracy
Large Whole-Proof Provers			
Kimina-Prover-Preview [Wang et al., 2025]	72B	8192	80.74%
DeepSeek-Prover-V2 (non-CoT) [Ren et al., 2025]	671B	8192	78.30%
DeepSeek-Prover-V2 (CoT) [Ren et al., 2025]	671B	8192	**88.90%**
Small Whole-Proof Provers			
Leanabell-Prover-GD-RL [Zhang et al., 2025]	7B	128	61.1%
Goedel-Prover-SFT [Lin et al., 2025]	7B	25600	64.7%
STP [Dong and Ma, 2025]	7B	25600	67.6%
Kimina-Prover-Preview-Distill [Wang et al., 2025]	7B	1024	70.8%
DeepSeek-Prover-V2 (Distilled, non-CoT) [Ren et al., 2025]	7B	8192	75.0%
DeepSeek-Prover-V2 (Distilled, CoT) [Ren et al., 2025]	7B	8192	82.0%
Step-level Provers			
InternLM2.5-StepProver + BFS + CG [Wu et al., 2024]	7B	$256 \times 32 \times 600$	65.9%
HunyuanProver + BFS + DC [Li et al., 2024b]	7B	$600 \times 8 \times 400$	68.4%
BFS-Prover [Xin et al., 2025]	7B	$2048 \times 2 \times 600$	70.83%
BFS-Prover [Xin et al., 2025]	7B	Accumulative	72.54%
MPS-Prover (Ours)	7B	$1 \times 4 \times 800 \times 8$	67.62%
		$4 \times 4 \times 800 \times 8$	68.44%
		$16 \times 4 \times 800 \times 8$	70.08%
		$64 \times 4 \times 800 \times 8$	72.54%
		Accumulative	**75.82%**

often yield performance exceeding that of models trained natively at the smaller scale [Guo et al., 2025]. In contrast, our model is trained directly via iterative refinement at the 7B scale. This comparison highlights the strong performance achieved by our method and suggests substantial potential for further improvement by leveraging larger base models or incorporating techniques like Chain-of-Thought (CoT) reasoning during tactic generation.

Another noteworthy finding is the strong performance of MPS-Prover even under constrained search budgets. At the minimum search budget evaluated, our model achieves a pass rate of 67.62% on miniF2F, significantly outperforming InternLM (50.7%) and Hunyuan Prover (59.84%) under similar minimal conditions. Impressively, this base performance already exceeds the maximum reported performance of several strong baselines, such as Goedel-prover and InternLM2.5-StepProver. This indicates that our approach exhibits excellent stability and efficiency, capable of achieving competitive results without necessitating exhaustive search efforts.

We also evaluate the performance of MPS-Prover on the ProofNet benchmark, which is generally considered more challenging than miniF2F, featuring undergraduate-level mathematics problems that demand more intricate reasoning. Table 3 presents a comparison of our method against other state-of-the-art 7B parameter models that have reported results on the ProofNet-test split. For a fair comparison, all models, including ours, were evaluated using their respective maximum reported sampling budgets. As can be observed, MPS-Prover achieves a success rate of 32.97%, surpassing all other 7B baseline models. Notably, our approach outperforms even DeepSeek-Prover-V2 with Chain-of-Thought (CoT) reasoning.

Table 3: ProofNet-test performance of different 7B models (max budget).

Method (7B models)	Performance
Goedel-Prover-SFT [Lin et al., 2025]	15.6%
STP [Dong and Ma, 2025]	26.9%
Deepseek-Prover-V1.5-RL [Xin et al., 2024]	25.3%
DeepSeek-Prover-V2 (non-CoT) [Ren et al., 2025]	24.7%
DeepSeek-Prover-V2 (CoT) [Ren et al., 2025]	29.6%
MPS-Prover (Ours)	**32.97%**

3.2 Comparison under Fixed Budgets

A crucial aspect of evaluating search algorithms is their performance relative to computational resources. Our Multi-Perspective Search (MPS) inherently explores more branches per iteration compared to standard Best-First Search (BFS) with tree-based distance prediction as critic [Li et al.,

2024b]. Specifically, MPS expands four nodes (one from the critic model and three from heuristic rules) for each selected state in a pass, whereas BFS typically expands only the single best node according to its criterion. Therefore, to ensure a fair comparison under approximately equivalent computational budgets, we compare the performance of MPS at pass@k against BFS at pass@4k.

As shown in Figure 3, MPS consistently outperforms BFS when allocated similar computational resources. At the lowest budget (MPS pass@1 vs. BFS pass@4), MPS achieves a success rate of 67.62% (165/244), slightly edging out BFS's 66.39% (162/244). This advantage becomes more pronounced as the budget increases. This consistent gap highlights the effectiveness of the diverse exploration strategy employed by MPS. By considering multiple perspectives (critic score + heuristics) at each step, MPS is less prone to getting stuck in local optima compared to the single-criterion approach of BFS, leading to a higher accuracy within a given computational budget.

3.3 Ablation Study

To understand the contribution of each component in our proposed MPS-Prover, we conduct a comprehensive ablation study. We evaluate the performance of our system by systematically removing or altering key components while keeping the total computational budget fixed. This budget is equivalent to our full method's pass@64 setting (i.e., $64 \times 4 \times 800 \times 8$). The experiments are performed on the miniF2F benchmark, and the results are presented in Table 4.

The ablation study reveals several key insights. First, removing our post-training data curation strategy results in a marginal decrease in performance, with our model solving one fewer problem. Thus, it offers substantial savings in training time and computational resources (40% less training data) with a negligible impact on the final proving capability. Second, when the critic model's score-based guidance is replaced with random selection of nodes for expansion, there is a drastic performance degradation (e.g., from 177 to 164 problems solved). This indicates that the learned critic model is effective in navigating the vast search space and guiding the prover towards promising proof paths. Finally, ablating each of our three heuristic rules for multi-perspective search by replacing their specific selection criteria with random choices also results in noticeable performance drops. In these variations ("w/o Tactic Eff. Heuristic", "w/o Min. Cases Heuristic", "w/o Short. State Heuristic"), the specific heuristic rule is deactivated, and its slot in the multi-perspective selection is filled by a random choice from the $N_{samples}$ candidate next states generated by the LLM for the current expansion. This demonstrates that without these heuristic rules diversifying tactic selection and guiding search, the prover is more susceptible to the inherent biases of the critic model alone. It becomes more likely to fall into local minima, explore unproductive tactic sequences, or even reach unprovable states.

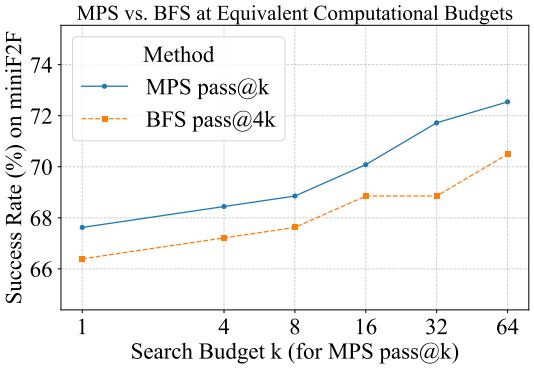


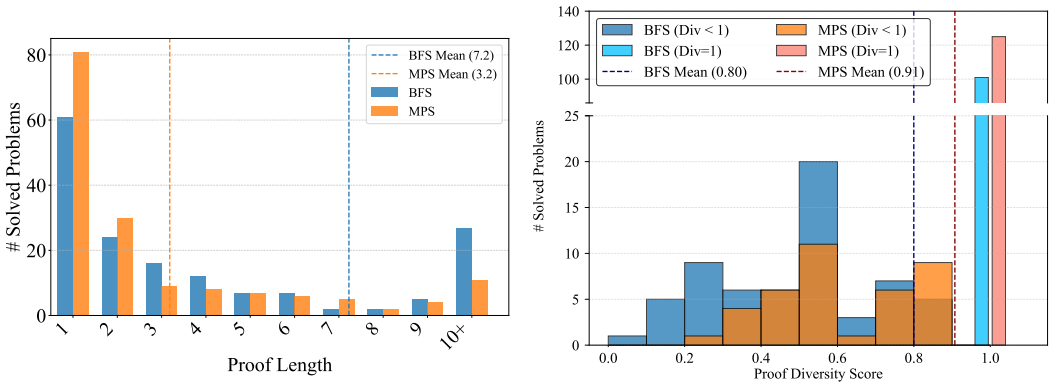
Figure 3: Performance comparison under equivalent computational budgets.

Table 4: Ablation study on miniF2F. Performance is problems proved (out of 244) under a fixed budget (MPS pass@64 equivalent). "w/o" indicates removing the specified component or replacing its guidance with random selection for heuristics.

Method Variation	Proved (n/244)
MPS-Prover (Full Method)	177/244
w/o Data Curation	176/244
w/o Critic Model	164/244
w/o Tactic Eff.	174/244
w/o Min. Cases	173/244
w/o Short. State	172/244

3.4 Proof Length and Diversity Analysis

To further investigate the characteristics of the proofs generated by different search strategies, we conduct a quantitative comparison between our Multi-Perspective Search (MPS) and standard Best-First Search (BFS) with tree-based distance prediction as critic. We ensure a fair comparison by using the identical LLM backbone and analyzing only the set of problems successfully proven by *both* MPS and BFS, guaranteeing analysis on the same theorems.



(a) Proof length distribution (steps) for commonly solved problems. Dashed lines indicate mean length.

(b) Proof diversity score distribution for commonly solved problems. Dashed lines indicate mean score.

Figure 4: Quantitative analysis of proof characteristics for commonly solved problems by BFS (pass@256) and MPS (pass@64).

Figure 4a shows the distribution of proof lengths, measured as the number of tactic steps (grouped into categories 1-9 and 10+). It is evident that proofs generated by MPS are significantly shorter on average than those found by BFS, as indicated by the mean values (dashed lines). MPS produces a higher frequency of proofs with fewer steps, while BFS exhibits a longer tail of lengthier proofs. This suggests that the diverse guidance signals in MPS help avoid unproductive tactic sequences or local optima, leading to more concise solutions.

Figure 4b illustrates the distribution of proof tactic diversity. We define diversity as the number of unique tactics used in a proof divided by its total length (number of steps). A score closer to 1 indicates a wider variety of tactics relative to length. The results clearly show that MPS-generated proofs possess considerably higher average diversity scores compared to BFS proofs (see mean lines). While both methods generate proofs with maximal diversity (score = 1.0, detailed in the annotation), BFS yields a much larger proportion of proofs with very low diversity scores. This highlights MPS's effectiveness in promoting exploration and leveraging a broader range of tactics, whereas BFS, guided solely by the critic model, is more prone to repetitive tactic usage.

3.5 MPS-Prover vs. Whole-Proof Provers: Proof Length

We further analyze proof characteristics by comparing the length of proofs generated by our MPS-Prover against two leading whole-proof provers, Kimina-Prover-Preview and DeepSeek-Prover-V2. This comparison uses 170 commonly solved miniF2F problems and measures proof length in Lean tactic steps. Table 5 reveals that MPS-Prover generates substantially shorter proofs (mean length 3.44 steps) compared to Kimina (15.91) and DeepSeek-Prover-V2 (52.16). Some examples of their proofs can be found in our Appendix C.

Table 5: Proof length statistics (Lean steps) on 170 common miniF2F problems.

Statistic	DeepSeek-V2	Kimina	MPS-Prover
Min	3	1	1
Max	698	186	37
Mean	52.16	15.91	3.44
Median	33.0	6.0	2.0
Std Dev	66.47	24.19	4.64

We attribute this to the operational differences: stepwise provers like MPS-Prover benefit from frequent interactions with the Lean engine. Each tactic execution updates the proof state, allowing the prover to adaptively refine its strategy by treating the new state as a sub-problem. This iterative process, combined with tactic-level search prioritizing impactful steps, facilitates the discovery of more direct solutions. In contrast, whole-proof systems often plan the entire proof initially, with limited dynamic adaptation, potentially leading to longer, albeit correct, proof scripts. We believe this analysis highlights a key advantage of step provers in producing more efficient proofs.

4 Related Work

Earlier Methods in Automated Theorem Proving Early automated theorem provers relied on symbolic search algorithms and hand-crafted heuristics. Systems like Vampire (a first-order logic

prover) [Riazanov and Voronkov, 2001] and SMT solvers like Z3 [de Moura and Bjørner, 2008] achieved impressive results using resolution, paramodulation, and DPLL-based search without learning. Some researchers apply premise selection in learning. Given a large library of axioms or lemmas, the goal is to predict which ones are relevant to a new theorem. Irving et al. [2016] pioneered deep learning for premise selection with the DeepMath system, using sequence models to rank premises for a target theorem. Similarly, Wang et al. [2017] used graph embeddings of knowledge bases to select premises, treating theorem proving as a graph traversal problem. Neural networks were also used to guide proof search directly inside automated provers. Loos et al. [2017] integrated a deep network into the E theorem prover, training the network to perform prediction.

LLM-based Whole-Proof Methods. Recent advances have seen LLMs directly generating complete formal proofs without iterative search, exploiting their powerful sequence modeling capabilities. Early examples include the Draft, Sketch, and Prove (DSP) system [Jiang et al., 2023], which uses informal natural language proofs as guidance, significantly improving prover accuracy; Baldur [First et al., 2023] generates proofs for Isabelle theorems and employs a repair mechanism leveraging failure feedback, achieving state-of-the-art results. LEGO-Prover [Wang et al., 2024b] enhances whole-proof generation by hierarchically proving and reusing lemmas, effectively managing intermediate results. Similarly, POETRY [Wang et al., 2024a] employs recursive proof decomposition, systematically breaking complex theorems into solvable subgoals. Additionally, curriculum learning strategies [Polu et al., 2022] and reinforcement learning [Dong et al., 2024, Xin et al., 2024] have been employed to optimize LLM performance. Goedel-Prover [Lin et al., 2025] and leanabell [Zhang et al., 2025] perform continual training with cognitive behavior data and RL outcomes from Lean 4 compiler. Kimina-Prover [Wang et al., 2025] demonstrates superior results (80.7% on miniF2F pass@8192) through structured reasoning patterns and RL training. DeepSeek-Prover-V2 [Ren et al., 2025] is trained via a recursive subgoal decomposition pipeline using DeepSeek-V3. By integrating CoT-style reasoning with formal proving, it achieves an impressive 88.9% accuracy on miniF2F.

LLM-based Step-level Tactic Generation Methods. Stepwise methods integrate LLMs into iterative proof searches, proposing individual proof steps and navigating search trees. GPT-f [Polu and Sutskever, 2020] pioneered this approach, proposing tactics that are verified incrementally, laying the groundwork for subsequent systems. HyperTree Proof Search (HTPS) [Lample et al., 2022] utilized an AlphaZero-inspired algorithm, exploring multiple proof branches simultaneously, significantly outperforming earlier methods through sophisticated search heuristics. LeanDojo's ReProver [Yang et al., 2023] incorporates premise retrieval, selecting relevant lemmas at each proof step, enhancing efficiency on Lean benchmarks. SubgoalXL [Zhao et al., 2024] employs expert-guided iterative training, optimizing subgoal generation strategies. ProofAug [Liu et al., 2025] further develops hybrid integration by alternately invoking neural suggestions, symbolic ATP calls, and recursive prover applications for efficient verification. Recent models like InternLM2.5-StepProver [Wu et al., 2024] utilized expert iteration with large-scale datasets. HunyuanProver [Li et al., 2024b] further enhanced data synthesis and guided tree search algorithms. BFS-Prover [Xin et al., 2025] demonstrated the efficacy of simpler Best-First Search methods, incorporating direct preference optimization from compiler feedback, and length normalization.

5 Discussion

In this work, we present the Multi-Perspective Search Prover (MPS-Prover), a novel stepwise automated theorem proving system that significantly advances the state of the art. By introducing a principled post-training data curation strategy and a multi-perspective tree search mechanism enhanced with heuristic critics, MPS-Prover effectively addresses common failure modes in existing stepwise provers, such as biased search and exploration of unproductive proof paths. Our extensive experiments demonstrate that MPS-Prover not only achieves superior success rates on challenging benchmarks like miniF2F and ProofNet but also generates proofs that are more concise and diverse.

For the broader ATP research community, our findings comfirms the strengths of stepwise proving, particularly in generating efficient proofs. Looking ahead, several promising avenues for future work emerge. One key direction is the development of hybrid systems that integrate the global planning capabilities of whole-proof methods with the adaptive, fine-grained search of stepwise provers like MPS-Prover. Another exciting prospect involves combining MPS-Prover with reinforcement learning (RL) techniques to further refine the critic model and search heuristics from self-play or direct feedback from the proof assistant.

References

DeepMind AlphaProof and AlphaGeometry Teams. Ai achieves silver-medal standard solving international mathematical olympiad problems.'25 july 2024, 2024.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Leonardo Mendonça de Moura and Nikolaj S Bjørner. Proofs and refutations, and z3. In *LPAR Workshops*, volume 418, pages 123–132. Doha, Qatar, 2008.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv e-prints*, pages arXiv–2502, 2025.

Kefan Dong, Arvind Mahankali, and Tengyu Ma. Formal theorem proving by rewarding llms to decompose proofs hierarchically. *arXiv preprint arXiv:2411.01829*, 2024.

Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath-deep sequence models for premise selection. *Advances in neural information processing systems*, 29, 2016.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023.

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35:26337–26349, 2022.

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

Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024b.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*, 2025.

Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *arXiv preprint arXiv:1701.06972*, 2017.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. Leveraging online olympiad-level math problems for llms training and contamination-resistant evaluation. *arXiv preprint arXiv:2501.14275*, 2025.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 625–635. Springer, 2021.

Lawrence C Paulson. *Isabelle: A generic theorem prover*. Springer, 1994.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

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.

Alexandre Riazanov and Andrei Voronkov. Vampire 1.1. In *Automated Reasoning: First International Joint Conference, IJCAR 2001 Siena, Italy, June 18–22, 2001 Proceedings 1*, pages 376–380. Springer, 2001.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, et al. Proving theorems recursively. *arXiv preprint arXiv:2405.14414*, 2024a.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, et al. Lego-prover: Neural theorem proving with growing libraries. In *12th International Conference on Learning Representations (ICLR 2024)*. International Conference on Learning Representations, ICLR, 2024b.

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

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. *Advances in neural information processing systems*, 30, 2017.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theoremllama: Transforming general-purpose llms into lean4 experts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11953–11974, 2024c.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

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. *arXiv preprint arXiv:2502.03438*, 2025.

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

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.

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

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

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *ICLR*, 2022.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the contributions (data curation, MPS-Prover, SOTA results, proof characteristics) which are supported by the experimental results presented in Section 4 (Experiments).

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Conclusion section briefly acknowledges the performance of larger models and discusses future work, implicitly noting current scope. A dedicated limitations section could be added if more depth is required by reviewers.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: This paper focuses on empirical contributions and novel system design rather than new theoretical results or mathematical proofs of algorithmic properties.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 (Experiments) details benchmarks, model training (SFT parameters, data composition), evaluation setup (Lean version, timeouts, search budget calculation), and specific data curation rules, providing a basis for reproducibility.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and instructions for reproducing experiments using publicly available benchmarks will be provided in supplemental material / upon publication via a public repository upon acceptance.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.2 (Supervised Fine-tuning) details training hyperparameters (learning rate, scheduler, epochs, batch size) and Section 4.3 (Evaluation Setup) describes Lean version, REPL usage, timeouts, and search budget components. Further details on data splits are implicit via standard benchmark usage.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: The paper reports absolute success rates (pass@k) on deterministic benchmarks. Due to the high computational cost of full ATP evaluation runs, multiple seeds for training or extensive runs for error bars were not performed. Comparisons are based on clear margins over SOTA.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.2 (Supervised Fine-tuning) mentions H20 80G GPUs and 3 days training.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research focuses on algorithmic improvements for automated theorem proving, a foundational AI task, and does not involve human subjects, sensitive data, or direct applications with immediate ethical concerns outlined in the Code of Ethics.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper focuses on foundational research in ATP. While improved ATP has positive implications for formal verification and mathematics, a detailed discussion of broader societal impacts was not included due to space and the fundamental nature of the work. Potential negative impacts are very indirect.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The research uses an LLM for tactic generation in a specialized domain (formal mathematics). While based on LLMs, the direct output (Lean tactics) does not pose a high risk for common misuse scenarios like disinformation or fake profile generation.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses publicly available benchmarks (miniF2F, ProofNet) and software (Lean, Lean REPL), which are cited. URLs for benchmark data sources are provided in footnotes (Section 4.1, 4.3). Licenses for these standard assets are generally known or available from their sources.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper introduces a new method (MPS-Prover) and a fine-tuned model but does not release new, standalone datasets or benchmarks. If model weights are released, documentation will accompany them.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The research does not involve crowdsourcing or direct human subject experimentation.

    Guidelines:

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The research does not involve human subjects, so IRB approval is not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The core methodology involves fine-tuning and using an LLM for tactic generation in automated theorem proving. This is explicitly described in Section 3 (Method) and Section 4 (Experiments).

A Limitation

While our MPS-Prover demonstrates significant advancements in stepwise automated theorem proving, it is important to acknowledge certain inherent limitations of the stepwise paradigm itself, particularly when compared to whole-proof generation approaches.

A primary limitation of current stepwise provers, including MPS-Prover, lies in their handling of tactics that introduce complex, nested proof obligations, such as new lemmas that require their own sub-proofs. Whole-proof systems, like DeepSeek-Prover-V2 [Guo et al., 2025], can generate entire proof scripts that include $have$ statements to introduce and subsequently prove auxiliary lemmas within the main proof structure. The verifier then processes the complete script. However, for a purely stepwise prover interacting with the Lean 4 engine, if a tactic attempts to introduce an unproven lemma or a complex structure requiring an immediate, unfulfilled sub-proof (e.g., via $have$, or certain intricate uses of $induction$ or $calc$ blocks that don't immediately resolve to simpler goals), the Lean engine will typically raise an error and halt that proof path. The prover cannot easily "pause" the main proof, prove the lemma in isolation, and then resume, all within a single interactive step.

This means that MPS-Prover, like other current step-provers, is less adept at autonomously discovering and utilizing complex intermediate lemmas that are not already present in the context or standard libraries. While our multi-perspective search can find efficient paths using existing tactics, it does not inherently support the generation and in-line proving of new, non-trivial lemmas in the same way a whole-proof generator might plan for. This restricts the prover's ability to break down very complex problems into more manageable, lemma-dependent sub-problems in a self-contained manner during the stepwise search.

Addressing this limitation is a key direction for future work. As mentioned in our conclusion, exploring hybrid approaches that combine the stepwise search capabilities of MPS-Prover with the global planning and lemma-handling strengths of whole-proof generation methods could offer a promising path towards overcoming this challenge and further expanding the scope of theorems that can be automatically proven.

B Baselines

We compare MPS-Prover against a comprehensive set of state-of-the-art automated theorem provers. For whole-proof generation methods, we include Kimina-Prover-Preview [Wang et al., 2025], which employs interleaved natural language reasoning and Lean code blocks along with reinforcement learning (RL). Another strong contender is DeepSeek-prover V2 [Ren et al., 2025], notable for its use of subgoal decomposition to break down complex problems and subsequent proof generation, also enhanced with RL. Goedel prover [Lin et al., 2025] represents methods focused on extensive data collection, having curated a large formalized mathematics dataset for expert iteration training. Leanabell-Prover [Zhang et al., 2025] similarly combines expert iteration with RL techniques. Additionally, STP [Dong and Ma, 2025] utilizes a dual-role architecture with a conjecturer and a prover, where each component provides training signals for the other. For stepwise proof generation methods, we select InternLM-StepProver [Wu et al., 2024], one of the pioneers in applying LLMs to step-level ATP. Hunyuan prover [Li et al., 2024b] advanced this line by designing improved critic models and integrating Monte Carlo Tree Search (MCTS). The most recent and leading baseline in this category is BFS-prover [Xin et al., 2025], which combines Supervised Fine-tuning (SFT) with Direct Preference Optimization (DPO) and incorporates length normalization during its Best-First Search, representing the previous state-of-the-art for step-provers.

C Case Studies

To provide a more nuanced understanding of the differences in proof strategies and generated solutions, we conduct case studies on specific theorems. We compare proofs generated by our MPS-Prover with those from Kimina-Prover and DeepSeek-Prover V2 for two commonly solved problems, and additionally showcase a problem uniquely solved by MPS-Prover.

C.1 Analysis of a Commonly Solved Problem: `algebra_absapbon1pabsapbleqsumabsaon1pabsa`

The theorem `algebra_absapbon1pabsapbleqsumabsaon1pabsa` states that for any real numbers a and b, $\frac{|a+b|}{1+|a+b|} \leq \frac{|a|}{1+|a|} + \frac{|b|}{1+|b|}$. All three provers successfully found a proof for this theorem, but their approaches and the resulting proof scripts differ significantly in length and style.

Our MPS-Prover generates a remarkably concise proof of only 8 lines. Key steps include leveraging rw for rewriting goals based on non-negativity, using by_cases for case analysis (e.g., $a = 0$), and then efficiently using $field\_simp$ with relevant hypotheses like abs_nonneg and mul_nonneg. The proof concludes with a call to $refine'$ combined with div_nonneg and powerful finishers like $nlinarith$ and $positivity$. Each step appears to make substantial progress, often simplifying the goal significantly or discharging parts of it by effectively utilizing built-in Mathlib lemmas and tactics. This conciseness stems from MPS-Prover's ability to explore and select tactics that yield significant progress at each step, guided by the multi-perspective search.

In contrast, the solution from Kimina-Prover for the same problem is considerably longer, spanning approximately 40 lines. It primarily relies on a sequence of $have$ statements to introduce intermediate lemmas (e.g., $h1 : abs(a+b) \leq absa+absb, h2 : \ldots, h3 : \ldots$). Each of these lemmas is then proven using a combination of more granular tactics like $apply$, $linarith$, $nlinarith$, and $field\_simp$. While logically sound, this approach of explicitly stating and proving multiple intermediate steps results in a more verbose proof. This suggests a strategy that might break down the problem based on an initial plan but does not necessarily find the most direct path available through more powerful, combined tactics.

The proof generated by DeepSeek-Prover V2 is the longest, around 60 lines. Similar to Kimina-Prover, it also extensively uses $have$ to introduce and prove a series of lemmas. However, the proof of those lemmas, and its subsequent applications, are broken down into many fine-grained steps involving $positivity$, $nlinarith$, $linarith$, and $rw[div\_le\_div\_iff \ldots]$. While demonstrating a sophisticated understanding by identifying and proving this general property, the overall proof becomes quite lengthy due to the detailed, step-by-step derivation of each component. This is characteristic of whole-proof systems that might formulate a high-level plan (e.g., "prove and use the monotonicity of $f(x) = x/(1+x)$") and then meticulously fill in the details, which may not always be the most compact way to express the proof in Lean.

These comparisons highlight that MPS-Prover, through its stepwise search and selection of high-progress tactics, tends to produce more concise proofs by effectively leveraging powerful built-in Mathlib functionalities. Whole-proof systems, while capable of complex reasoning and planning, may generate longer proofs due to their tendency to break down problems into many simpler, explicitly stated intermediate steps, potentially constrained by an initial, possibly more verbose, plan.

```
-- Solution to algebra_absapbon1pabsapbleqsumabsaon1pabsa by our MPS-Prover

import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

theorem algebra_absapbon1pabsapbleqsumabsaon1pabsa (a b : ℝ) :
    abs (a + b) / (1 + abs (a + b)) ≤ abs a / (1 + abs a) + abs b / (1 + abs b) :=
    ↪ by

  rw [← sub_nonneg, ← sub_nonneg, ← sub_nonneg]
  by_cases h₁ : a = 0
  all_goals field_simp [abs_of_nonneg, abs_of_pos, *]
  have h₂ : 0 ≤ |b| := abs_nonneg b
<;> have h₃ : 0 ≤ |a| := abs_nonneg a
<;> field_simp [h₁, h₂, h₃]
  have h₄ : 0 ≤ |a + b| := abs_nonneg (a + b)
<;>
have h₅ : 0 ≤ |a| * |b| := mul_nonneg h₃ h₂
<;>
```

```
field_simp [h₁, h₂, h₃, h₄, h₅]
  refine' div_nonneg (by nlinarith [abs_mul a b, abs_add a b]) (by positivity)
```

```
-- Solution to algebra_absapbon1pabsapbleqsumabsaon1pabsa by Kimina-Prover

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem algebra_absapbon1pabsapbleqsumabsaon1pabsa (a b : ℝ) :
    abs (a + b) / (1 + abs (a + b)) ≤ abs a / (1 + abs a) + abs b / (1 + abs b) :=
    ↪ by
  have h1 : abs (a + b) ≤ abs a + abs b := by
    apply abs_add
  have h2 : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + abs a + abs
  ↪ b) := by
    have h2a : abs (a + b) ≤ abs a + abs b := by
      apply abs_add
    have h2b : 0 ≤ abs (a + b) := abs_nonneg (a + b)
    have h2c : 0 ≤ abs a + abs b := by
      apply add_nonneg
      · apply abs_nonneg a
      · apply abs_nonneg b
    have h2d : 0 ≤ (1 + abs (a + b)) := by
      linarith [abs_nonneg (a + b)]
    have h2e : 0 ≤ (1 + abs a + abs b) := by
      linarith [abs_nonneg a, abs_nonneg b]
    have h2f : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + abs a +
    ↪ abs b) := by
      apply (div_le_div_iff (by linarith) (by linarith)).mpr
      nlinarith [abs_nonneg (a + b), abs_nonneg a, abs_nonneg b, h2a]
    exact h2f
  have h3 : (abs a + abs b) / (1 + abs a + abs b) ≤ abs a / (1 + abs a) + abs b /
  ↪ (1 + abs b) := by
    have h3a : 0 ≤ (1 + abs a + abs b) := by
      linarith [abs_nonneg a, abs_nonneg b]
    have h3b : 0 ≤ (1 + abs a) := by
      linarith [abs_nonneg a]
    have h3c : 0 ≤ (1 + abs b) := by
      linarith [abs_nonneg b]
    have h4 : (abs a + abs b) / (1 + abs a + abs b) ≤ abs a / (1 + abs a) + abs b
    ↪ / (1 + abs b) := by
      have h4a : abs a ≥ 0 := abs_nonneg a
      have h4b : abs b ≥ 0 := abs_nonneg b
      have h4c : 1 + abs a + abs b > 0 := by linarith [abs_nonneg a, abs_nonneg b]
      have h4d : 1 + abs a > 0 := by linarith [abs_nonneg a]
      have h4e : 1 + abs b > 0 := by linarith [abs_nonneg b]
      have h5 : abs a / (1 + abs a) + abs b / (1 + abs b) - (abs a + abs b) / (1 +
      ↪ abs a + abs b) ≥ 0 := by
        have h5a : abs a / (1 + abs a) + abs b / (1 + abs b) - (abs a + abs b) /
        ↪ (1 + abs a + abs b) =
          (abs a * (1 + abs b) * (1 + abs a + abs b) +
          abs b * (1 + abs a) * (1 + abs a + abs b) -
          (abs a + abs b) * (1 + abs a) * (1 + abs b)) /
          ((1 + abs a) * (1 + abs b) * (1 + abs a + abs b)) := by
          field_simp
          <;> ring
        rw [h5a]
        have h5b : 0 ≤ (abs a * (1 + abs b) * (1 + abs a + abs b) +
```

```
          abs b * (1 + abs a) * (1 + abs a + abs b) -
          (abs a + abs b) * (1 + abs a) * (1 + abs b)) := by
          nlinarith [sq_nonneg (abs a), sq_nonneg (abs b), mul_nonneg (abs_nonneg
          ↪  a) (abs_nonneg b)]
        have h5c : 0 < (1 + abs a) * (1 + abs b) * (1 + abs a + abs b) := by
          apply mul_pos
          apply mul_pos
          · linarith [abs_nonneg a]
          · linarith [abs_nonneg b]
          · linarith [abs_nonneg a, abs_nonneg b]
        apply div_nonneg
        · linarith
        · nlinarith [abs_nonneg a, abs_nonneg b]
      linarith [h5]
    linarith [h4]
  linarith [h2, h3]
```

```
-- Solution to algebra_absapbon1pabsapbleqsumabsaon1pabsa by Deepseek-Prover V2

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Show that for any real numbers $a$ and $b$, $\frac{|a+b|}{1+|a+b|}\leq
↪  \frac{|a|}{1+|a|}+\frac{|b|}{1+|b|}$.-/
theorem algebra_absapbon1pabsapbleqsumabsaon1pabsa (a b : ℝ) :
    abs (a + b) / (1 + abs (a + b)) ≤ abs a / (1 + abs a) + abs b / (1 + abs b) :=
    ↪  by
  have h₁ : abs (a + b) ≤ abs a + abs b := by
    calc
      abs (a + b) ≤ abs a + abs b := by
        -- Use the triangle inequality for absolute values.
        exact abs_add a b
      _ = abs a + abs b := by rfl

  have h₂ : ∀ (x y : ℝ), 0 ≤ x → x ≤ y → x / (1 + x) ≤ y / (1 + y) := by
    intro x y hx hxy
    have h₃ : 0 ≤ x := hx
    have h₄ : x ≤ y := hxy
    have h₅ : 0 ≤ y := by linarith
    have h₆ : 0 ≤ 1 + x := by linarith
    have h₇ : 0 ≤ 1 + y := by linarith
    have h₈ : 0 < 1 + x := by linarith
    have h₉ : 0 < 1 + y := by linarith
    -- Use the division inequality to compare the fractions.
    have h₁₀ : x / (1 + x) ≤ y / (1 + y) := by
      rw [div_le_div_iff (by positivity) (by positivity)]
      -- Simplify the inequality to a form that can be handled by linear
      ↪  arithmetic.
      nlinarith [mul_nonneg h₃ (sub_nonneg.mpr h₄), mul_nonneg h₅ h₃,
        mul_nonneg h₅ (sub_nonneg.mpr h₄)]
    exact h₁₀

  have h₃ : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + (abs a + abs
  ↪  b)) := by
    have h₄ : 0 ≤ abs (a + b) := abs_nonneg (a + b)
    have h₅ : 0 ≤ abs a + abs b := by positivity
    have h₆ : abs (a + b) ≤ abs a + abs b := h₁
    have h₇ : 0 ≤ 1 + abs (a + b) := by positivity
```

```
    have h₈ : 0 ≤ 1 + (abs a + abs b) := by positivity
    have h₉ : 0 < 1 + abs (a + b) := by positivity
    have h₁₀ : 0 < 1 + (abs a + abs b) := by positivity
    -- Use the fact that the function x / (1 + x) is increasing for x ≥ 0.
    have h₁₁ : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + (abs a +
    ↪  abs b)) := by
      have h₁₂ : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + (abs a +
      ↪  abs b)) := by
        -- Apply the increasing property of the function.
        have h₁₃ : 0 ≤ abs (a + b) := by positivity
        have h₁₄ : abs (a + b) ≤ abs a + abs b := h₁
        have h₁₅ : 0 ≤ abs a + abs b := by positivity
        have h₁₆ : 0 < 1 + abs (a + b) := by positivity
        have h₁₇ : 0 < 1 + (abs a + abs b) := by positivity
        -- Use the previously proven lemma.
        have h₁₈ : abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + (abs a
        ↪  + abs b)) := by
          apply h₂
          <;> try norm_num
          <;> linarith
        exact h₁₈
      exact h₁₂
    exact h₁₁

  have h₄ : (abs a + abs b) / (1 + (abs a + abs b)) ≤ abs a / (1 + abs a) + abs b
  ↪  / (1 + abs b) := by
    have h₅ : 0 ≤ abs a := abs_nonneg a
    have h₆ : 0 ≤ abs b := abs_nonneg b
    have h₇ : 0 ≤ abs a + abs b := by positivity
    have h₈ : 0 < 1 + abs a := by positivity
    have h₉ : 0 < 1 + abs b := by positivity
    have h₁₀ : 0 < 1 + (abs a + abs b) := by positivity
    have h₁₁ : (abs a + abs b) / (1 + (abs a + abs b)) = abs a / (1 + (abs a + abs
    ↪  b)) + abs b / (1 + (abs a + abs b)) := by
      field_simp [h₈, h₉, h₁₀]
      <;> ring
      <;> field_simp [h₈, h₉, h₁₀]
      <;> ring
    rw [h₁₁]
    have h₁₂ : abs a / (1 + (abs a + abs b)) ≤ abs a / (1 + abs a) := by
      have h₁₃ : 0 ≤ abs a := abs_nonneg a
      have h₁₄ : 0 < 1 + abs a := by positivity
      have h₁₅ : 0 < 1 + (abs a + abs b) := by positivity
      have h₁₆ : 0 < 1 + abs a + abs b := by positivity
      -- Use the fact that 1 + abs a ≤ 1 + (abs a + abs b) to show the inequality
      have h₁₇ : 1 + abs a ≤ 1 + (abs a + abs b) := by
        nlinarith [h₆]
      -- Use the division inequality to prove the result
      have h₁₈ : abs a / (1 + (abs a + abs b)) ≤ abs a / (1 + abs a) := by
        apply div_le_div_of_le_left (by positivity) (by positivity)
        <;> nlinarith
      exact h₁₈
    have h₁₃ : abs b / (1 + (abs a + abs b)) ≤ abs b / (1 + abs b) := by
      have h₁₄ : 0 ≤ abs b := abs_nonneg b
      have h₁₅ : 0 < 1 + abs b := by positivity
      have h₁₆ : 0 < 1 + (abs a + abs b) := by positivity
      have h₁₇ : 0 < 1 + abs a + abs b := by positivity
      -- Use the fact that 1 + abs b ≤ 1 + (abs a + abs b) to show the inequality
      have h₁₈ : 1 + abs b ≤ 1 + (abs a + abs b) := by
        nlinarith [h₅]
      -- Use the division inequality to prove the result
      have h₁₉ : abs b / (1 + (abs a + abs b)) ≤ abs b / (1 + abs b) := by
        apply div_le_div_of_le_left (by positivity) (by positivity)
        <;> nlinarith
```

```
      exact h₁₉
    have h₂₀ : abs a / (1 + (abs a + abs b)) + abs b / (1 + (abs a + abs b)) ≤ abs
    ↪  a / (1 + abs a) + abs b / (1 + abs b) := by
      linarith
    linarith

  have h₅ : abs (a + b) / (1 + abs (a + b)) ≤ abs a / (1 + abs a) + abs b / (1 +
  ↪  abs b) := by
    calc
      abs (a + b) / (1 + abs (a + b)) ≤ (abs a + abs b) / (1 + (abs a + abs b)) :=
      ↪  h₃
      _ ≤ abs a / (1 + abs a) + abs b / (1 + abs b) := h₄

  exact h₅
```

C.2 Analysis of a Commonly Solved Problem: `imo_1962_p2`

The theorem `imo_1962_p2` is another problem successfully solved by all three provers, allowing for a comparison of proof styles for a different type of problem involving inequalities and square roots. Our MPS-Prover again produces a very short proof. It effectively uses $refine'$ to structure the proof for the conjunction, then leverages a sequence of powerful rewriting and simplification tactics like rw, $field\_simp$, $norm\_num$, and $nlinarith$, often chained or applied with specific hypotheses. This demonstrates an ability to quickly simplify complex algebraic expressions and discharge goals using arithmetic reasoning.

Kimina-Prover's solution is also structured around proving the two conjuncts separately using $constructor$. It uses $have$ to establish intermediate inequalities and then applies tactics like $linarith$, sq_lt_sq', $rw[Real.sq\_sqrt]$, and $nlinarith$. The steps are logical and clear but involve more explicit intermediate assertions compared to MPS-Prover.

DeepSeek-Prover V2's proof is the most detailed. It also uses $constructor$ (implicitly, by proving h_3 and h_{11} separately for the conjuncts) and introduces many intermediate facts with $have$. The reasoning involves careful manipulation of inequalities, squaring both sides (e.g., $gcongr$), and algebraic simplification, often broken down into very small, verifiable steps. While demonstrating thoroughness, this level of granularity contributes to the proof's length.

This case again illustrates MPS-Prover's tendency towards conciseness by selecting powerful, progress-making tactics. The whole-proof methods, while effective, often produce more verbose proofs due to their strategy of explicit lemma introduction and detailed, step-by-step derivations.

```
-- Solution to imo_1962_p2 by our MPS-Prover

import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

theorem imo_1962_p2 (x : ℝ) (h₀ : 0 ≤ 3 - x) (h₁ : 0 ≤ x + 1)
    (h₂ : 1 / 2 < Real.sqrt (3 - x) - Real.sqrt (x + 1)) : -1 ≤ x ∧ x < 1 -
    ↪  Real.sqrt 31 / 8 := by

  refine' ⟨_, _⟩
  nlinarith [Real.sq_sqrt h₀, Real.sq_sqrt h₁]
  rw [lt_sub_iff_add_lt] at h₂
  rw [lt_sqrt (by positivity)] at h₂
  rw [← sub_pos] at h₂ ⊢
  ring_nf
  ring_nf at h₀ h₁ h₂ ⊢
  field_simp at h₂ ⊢
<;> ring_nf at h₂ ⊢ <;> norm_num [h₀, h₁, h₂]
  rw [Real.sqrt_lt (by linarith)] <;> norm_num
<;> nlinarith [Real.sqrt_nonneg (1 + x), Real.sq_sqrt h₁]
```

```
-- Solution to imo_1962_p2 by Kimina-Prover

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat


theorem imo_1962_p2 (x : ℝ) (h₀ : 0 ≤ 3 - x) (h₁ : 0 ≤ x + 1)
    (h₂ : 1 / 2 < Real.sqrt (3 - x) - Real.sqrt (x + 1)) : -1 ≤ x ∧ x < 1 -
    ↪ Real.sqrt 31 / 8 := by
  constructor
  · linarith [h₁]
  · have h3 : Real.sqrt (3 - x) - Real.sqrt (x + 1) > 1 / 2 := by linarith
    have h4 : Real.sqrt (3 - x) > Real.sqrt (x + 1) + 1 / 2 := by linarith
    have h5 : Real.sqrt (x + 1) ≥ 0 := Real.sqrt_nonneg (x + 1)
    have h6 : Real.sqrt (3 - x) ≥ 0 := Real.sqrt_nonneg (3 - x)
    have h7 : Real.sqrt (3 - x) > 0 := by linarith [h6, h5]
    have h8 : (Real.sqrt (3 - x)) ^ 2 > (Real.sqrt (x + 1) + 1 / 2) ^ 2 := by
      apply sq_lt_sq'
      all_goals linarith [h4, h5, h7]
    have h9 : (Real.sqrt (3 - x)) ^ 2 = 3 - x := by
      rw [Real.sq_sqrt]
      linarith
    have h10 : (Real.sqrt (x + 1) + 1 / 2) ^ 2 = (Real.sqrt (x + 1)) ^ 2 +
    ↪ Real.sqrt (x + 1) + 1 / 4 := by
      ring_nf
      <;> simp [Real.sqrt_nonneg]
    rw [h9, h10] at h8
    have h11 : (Real.sqrt (x + 1)) ^ 2 = x + 1 := by
      rw [Real.sq_sqrt]
      linarith
    have h12 : Real.sqrt (x + 1) < 7 / 4 - 2 * x := by nlinarith [h8, h11,
    ↪ Real.sqrt_nonneg (x + 1)]
    have h13 : x < 1 - Real.sqrt 31 / 8 := by
      have h14 : Real.sqrt (x + 1) + 2 * x < 7 / 4 := by linarith [h12]
      have h15 : Real.sqrt 31 ^ 2 = 31 := by
        rw [Real.sq_sqrt]
        norm_num
      nlinarith [h14, Real.sq_sqrt (show 0 ≤ x + 1 by linarith), Real.sq_sqrt
      ↪ (show 0 ≤ 3 - x by linarith), Real.sqrt_nonneg (3 - x), Real.sqrt_nonneg
      ↪ (x + 1), Real.sq_sqrt (show 0 ≤ (31 : ℝ) by norm_num), Real.sq_sqrt
      ↪ (show 0 ≤ (8 : ℝ) by norm_num)]
    linarith [h13]
```

```
-- Solution to imo_1962_p2 by Deepseek-Prover V2

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat


theorem imo_1962_p2 (x : ℝ) (h₀ : 0 ≤ 3 - x) (h₁ : 0 ≤ x + 1)
    (h₂ : 1 / 2 < Real.sqrt (3 - x) - Real.sqrt (x + 1)) : -1 ≤ x ∧ x < 1 -
    ↪ Real.sqrt 31 / 8 := by
```

```
  have h₃ : -1 ≤ x := by
    linarith

  have h₄ : Real.sqrt (3 - x) - Real.sqrt (x + 1) > 1 / 2 := by
    linarith

  have h₅ : Real.sqrt (3 - x) > Real.sqrt (x + 1) + 1 / 2 := by
    linarith

  have h₆ : 3 - x > x + 1 + Real.sqrt (x + 1) + 1 / 4 := by
    have h₆₁ : Real.sqrt (3 - x) > Real.sqrt (x + 1) + 1 / 2 := h₅
    have h₆₂ : 0 ≤ Real.sqrt (x + 1) := Real.sqrt_nonneg (x + 1)
    have h₆₃ : 0 ≤ Real.sqrt (3 - x) := Real.sqrt_nonneg (3 - x)
    have h₆₄ : 0 ≤ Real.sqrt (x + 1) * (1 / 2) := by positivity
    have h₆₅ : 0 ≤ Real.sqrt (3 - x) := Real.sqrt_nonneg (3 - x)
    have h₆₆ : 0 ≤ Real.sqrt (x + 1) := Real.sqrt_nonneg (x + 1)
    have h₆₇ : (Real.sqrt (x + 1) + 1 / 2) ≥ 0 := by positivity
    have h₆₈ : Real.sqrt (3 - x) > Real.sqrt (x + 1) + 1 / 2 := h₅
    have h₆₉ : (Real.sqrt (x + 1) + 1 / 2) ^ 2 < (Real.sqrt (3 - x)) ^ 2 := by
      gcongr
      <;> nlinarith [Real.sq_sqrt (show 0 ≤ x + 1 by linarith), Real.sq_sqrt (show
      ↪  0 ≤ 3 - x by linarith)]
    have h₆₁₀ : (Real.sqrt (x + 1) + 1 / 2) ^ 2 = (x + 1) + Real.sqrt (x + 1) + 1 /
    ↪  4 := by
      nlinarith [Real.sq_sqrt (show 0 ≤ x + 1 by linarith), Real.sq_sqrt (show 0
      ↪  ≤ 3 - x by linarith),
        sq_nonneg (Real.sqrt (x + 1) - 1 / 2)]
    have h₆₁₁ : (Real.sqrt (3 - x)) ^ 2 = 3 - x := by
      rw [Real.sq_sqrt] <;> linarith
    nlinarith [Real.sq_sqrt (show 0 ≤ x + 1 by linarith), Real.sq_sqrt (show 0 ≤
    ↪  3 - x by linarith)]

  have h₇ : 7 - 8 * x > 4 * Real.sqrt (x + 1) := by
    have h₇₁ : 3 - x > x + 1 + Real.sqrt (x + 1) + 1 / 4 := h₆
    have h₇₂ : 0 ≤ Real.sqrt (x + 1) := Real.sqrt_nonneg (x + 1)
    nlinarith [Real.sqrt_nonneg (x + 1)]

  have h₈ : x < 7 / 8 := by
    by_contra h
    have h₈₁ : x ≥ 7 / 8 := by linarith
    have h₈₂ : 7 - 8 * x ≤ 0 := by
      linarith
    have h₈₃ : 4 * Real.sqrt (x + 1) ≥ 0 := by positivity
    linarith

  have h₉ : 0 ≤ Real.sqrt 31 := by positivity

  have h₁₀ : 64 * x ^ 2 - 128 * x + 33 > 0 := by
    have h₁₀₁ : 7 - 8 * x > 4 * Real.sqrt (x + 1) := h₇
    have h₁₀₂ : 0 ≤ Real.sqrt (x + 1) := Real.sqrt_nonneg (x + 1)
    have h₁₀₃ : 0 ≤ 7 - 8 * x := by linarith
    have h₁₀₄ : 0 ≤ Real.sqrt (x + 1) := Real.sqrt_nonneg (x + 1)
    have h₁₀₅ : 0 ≤ 7 - 8 * x := by linarith
    have h₁₀₆ : (7 - 8 * x) ^ 2 > (4 * Real.sqrt (x + 1)) ^ 2 := by
      gcongr
    have h₁₀₇ : (4 * Real.sqrt (x + 1)) ^ 2 = 16 * (x + 1) := by
      nlinarith [Real.sq_sqrt (show 0 ≤ x + 1 by linarith), sq_nonneg (Real.sqrt
      ↪  (x + 1) - 2)]
    have h₁₀₈ : (7 - 8 * x) ^ 2 > 16 * (x + 1) := by
      linarith
    nlinarith [Real.sq_sqrt (show 0 ≤ x + 1 by linarith), sq_nonneg (x - 1)]

  have h₁₁ : x < 1 - Real.sqrt 31 / 8 := by
    by_contra h
```

```
    have h₁₁₁ : x ≥ 1 - Real.sqrt 31 / 8 := by
      linarith
    have h₁₁₂ : 64 * x ^ 2 - 128 * x + 33 ≤ 0 := by
      have h₁₁₃ : x ≥ 1 - Real.sqrt 31 / 8 := h₁₁₁
      have h₁₁₄ : x ≤ 1 + Real.sqrt 31 / 8 := by
        by_contra h₁₁₄
        have h₁₁₅ : x > 1 + Real.sqrt 31 / 8 := by linarith
        have h₁₁₆ : 64 * x ^ 2 - 128 * x + 33 > 0 := h₁₀
        have h₁₁₇ : x < 7 / 8 := h₈
        nlinarith [Real.sqrt_nonneg 31, Real.sq_sqrt (show 0 ≤ 31 by norm_num)]
      nlinarith [Real.sqrt_nonneg 31, Real.sq_sqrt (show 0 ≤ 31 by norm_num),
        sq_nonneg (x - (1 - Real.sqrt 31 / 8)), sq_nonneg (x - (1 + Real.sqrt 31 /
        ↪ 8))]
    linarith [h₁₀]

  exact ⟨h₃, h₁₁⟩
```

C.3 Analysis of a Problem Uniquely Solved by MPS-Prover: `amc12a_2020_p7`

The problem `amc12a_2020_p7` involves evaluating a sum based on a sequence defined by specific values of $a(k)^3$. This problem was solved by our MPS-Prover, while both Kimina-Prover and DeepSeek-Prover V2 failed to find a solution.

The MPS-Prover solution employs $induction'$ as a key initial step. This is followed by extensive use of $simp\_all$ (often with $config := \{decide := true\}$) to simplify goals after instantiating terms of the sequence ($a(0)$ through $a(6)$) derived from the hypotheses h_0 through h_6. The proof also uses $interval\_cases$ to handle natural number variables within certain ranges derived from $nlinarith$. The final steps involve further simplification with $Nat.cast$ properties. The success here suggests that MPS-Prover's search was able to identify a productive high-level strategy ($induction'$) and then effectively use simplification and case analysis tactics to manage the resulting subgoals. The ability to find this particular combination of tactics, especially the crucial $induction'$ and effective use of $interval\_cases$, within the search budget highlights the strength of our multi-perspective approach in navigating complex search spaces where other methods might falter. The other provers might have struggled to identify the correct induction variable or effectively simplify the numerous concrete arithmetic subgoals that arise.

This case study highlights MPS-Prover's ability to find solutions to problems that prove difficult for other leading systems. Such successes can be attributed to the robust search paradigm of stepwise provers, which allows for exploration of various proof strategies, selection of effective tactics, and crucial backtracking capabilities. These features, amplified by our multi-perspective enhancements, enable the discovery of solutions even when the reasoning path is intricate or non-obvious.

```
-- A problem solved by our MPS-Prover, where Kimina-Prover and Deepseek-Prover
↪  both fail to solve.

import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

theorem amc12a_2020_p7 (a : ℕ → ℕ) (h₀ : (a 0)^3 = 1) (h₁ : (a 1)^3 = 8) (h₂ : (a
↪  2)^3 = 27) (h₃ : (a 3)^3 = 64) (h₄ : (a 4)^3 = 125) (h₅ : (a 5)^3 = 216) (h₆ :
↪  (a 6)^3 = 343) : ∑ k in Finset.range 7, (6 * (a k)^2) - ↑(2 * ∑ k in
↪  Finset.range 6, (a k)^2) = 658 := by

  induction' 4 <;> simp_all [Finset.sum_range_succ, pow_succ]
  have h₇ : a 1 ≤ 8 := by nlinarith
  interval_cases a 1 <;> simp_all (config := {decide := true})
  have h₇ : a 2 * a 2 * a 2 = 27 := by assumption
  have h₈ : a 3 * a 3 * a 3 = 64 := h₃
<;> try simp_all [Nat.mul_comm, Nat.mul_assoc, Nat.mul_left_comm]
```

```
  have h₉ : a 4 * a 4 * a 4 = 125 := by nlinarith
  all_goals
  have : a 2 ≤ 6 := by nlinarith
  interval_cases a 2 <;> simp_all (config := {decide := true})
  have h₁₀ : a 5 * a 5 * a 5 = 216 := by nlinarith
  have h₁₂ : a 6 = 7 := by nlinarith
  simp_all
  all_goals
  have : a 4 ≤ 12 := by nlinarith
  interval_cases a 4 <;> simp_all (config := {decide := true})
  have h₁₄ : a 5 = 6 := by nlinarith
  simp_all [Nat.cast_add, Nat.cast_mul, Nat.cast_pow]
  have h₁₁ : a 3 ≤ 8 := by nlinarith
  interval_cases a 3 <;> simp_all
```