DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition

Z.Z. Ren*, Zhihong Shao*, Junxiao Song*, Huajian Xin†, Haocheng Wang†, Wanjia Zhao†, Liyue Zhang, Zhe Fu
Qihao Zhu, Dejian Yang, Z.F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao
Daya Guo, Chong Ruan

DeepSeek-AI

https://github.com/deepseek-ai/DeepSeek-Prover-V2

Abstract

We introduce DeepSeek-Prover-V2, an open-source large language model designed for formal theorem proving in Lean 4, with initialization data collected through a recursive theorem proving pipeline powered by DeepSeek-V3. The cold-start training procedure begins by prompting DeepSeek-V3 to decompose complex problems into a series of subgoals. The proofs of resolved subgoals are synthesized into a chain-of-thought process, combined with DeepSeek-V3's step-by-step reasoning, to create an initial cold start for reinforcement learning. This process enables us to integrate both informal and formal mathematical reasoning into a unified model. The resulting model, DeepSeek-Prover-V2-671B, achieves state-of-the-art performance in neural theorem proving, reaching 88.9% pass ratio on the MiniF2F-test and solving 47 out of 658 problems from PutnamBench. In addition to standard benchmarks, we introduce ProverBench, a collection of 325 formalized problems, to enrich our evaluation, including 15 selected problems from the recent AIME competitions (years 24-25). Further evaluation on these 15 AIME problems shows that the model successfully solves 6 of them. In comparison, DeepSeek-V3 solves 8 of these problems using majority voting, highlighting that the gap between formal and informal mathematical reasoning in large language models is substantially narrowing.

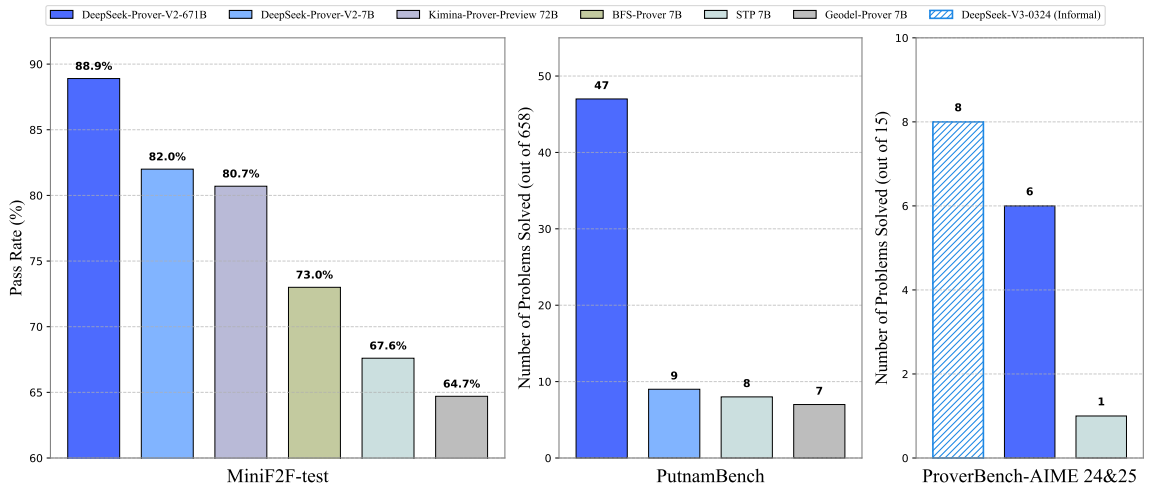

Figure 1 | Benchmark performance of DeepSeek-Prover-V2. On the AIME benchmark, DeepSeek-V3 is evaluated using the standard find-answer task for natural-language reasoning, while prover models generate Lean code to construct formal proofs for a given correct answer.

*Core contributors †Work done during internship at DeepSeek-AI.

1. Introduction

The emergence of reasoning capabilities in large language models (LLMs) has revolutionized numerous areas of artificial intelligence, particularly in the domain of mathematical problem solving (DeepSeek-AI, 2025). These advancements are largely enabled by the paradigm of inference-time scaling, most notably through natural language chain-of-thought reasoning (Jaech et al., 2024). Rather than relying solely on a single forward pass to arrive at an answer, LLMs can reflect on intermediate reasoning steps, improving both accuracy and interpretability. Despite the success of natural language reasoning in solving competition-level mathematical problems, its application to formal theorem proving remains fundamentally challenging. LLMs perform natural language reasoning in an inherently informal manner, relying on heuristics, approximations, and data-driven guessing patterns that often lack the rigorous structure required by formal verification systems. In contrast, proof assistants such as Lean (Moura and Ullrich, 2021), Isabelle (Paulson, 1994), and Coq (Barras et al., 1999) operate on strict logical foundations, where every proof step must be explicitly constructed and formally verified. These systems permit no ambiguity, implicit assumptions, or omission of details. Bridging the gap between informal, high-level reasoning and the syntactic rigor of formal verification systems remains a longstanding research challenge in neural theorem proving (Yang et al., 2024).

To harness the strengths of informal mathematical reasoning in support of formal theorem proving, a classical approach is to hierarchically decompose formal proofs based on the guidance of natural-language proof sketches. Jiang et al. (2023) proposed a framework, called *Draft, Sketch, and Prove* (DSP), that leverages a large language model to generate proof sketches in natural language, which are subsequently translated into formal proof steps. This informal-to-formal theorem proving paradigm closely mirrors the concept of subgoals in hierarchical reinforcement learning (Barto and Mahadevan, 2003; Nachum et al., 2018; Eppe et al., 2022), where complex tasks are broken down into a hierarchy of simpler subtasks that can be solved independently to progressively achieve the overarching objective. In formal theorem proving, a subgoal is typically an intermediate proposition or lemma that contributes to the proof of a larger theorem (Zhao et al., 2023, 2024). This hierarchical decomposition aligns with human problem-solving strategies and supports modularity, reusability, and more efficient proof search (Wang et al., 2024b; Zheng et al., 2024). Recent studies have extended this paradigm by employing multi-tiered hierarchies for structured proof generation (Wang et al., 2024a), and by leveraging reinforcement learning techniques to optimize the decomposition of complex theorems into manageable subgoals (Dong et al., 2024).

In this paper, we develop a reasoning model for subgoal decomposition, leveraging a suite of synthetic cold-start data and large-scale reinforcement learning to enhance its performance. To construct the cold-start dataset, we develop a simple yet effective pipeline for recursive theorem proving, utilizing DeepSeek-V3 (DeepSeek-AI, 2024) as a unified tool for both subgoal decomposition and formalization. We prompt DeepSeek-V3 to decompose theorems into high-level proof sketches while simultaneously formalizing these proof steps in Lean 4, resulting in a sequence of subgoals. Since the subgoal decomposition is powered by a large general-purpose model, we use a smaller 7B model to handle the proof search for each subgoal, thereby reducing the associated computational burden. Additionally, we introduce a curriculum learning framework that leverages the decomposed subgoals to generate conjectural theorems, progressively increasing the difficulty of training tasks to better guide the model's learning process. Once the decomposed steps of a challenging problem are resolved, we pair the complete step-by-step formal proof with the corresponding chain-of-thought from DeepSeek-V3 to create cold-start reasoning data. Based on the cold start, a subsequent reinforcement learning stage is applied to further strengthen

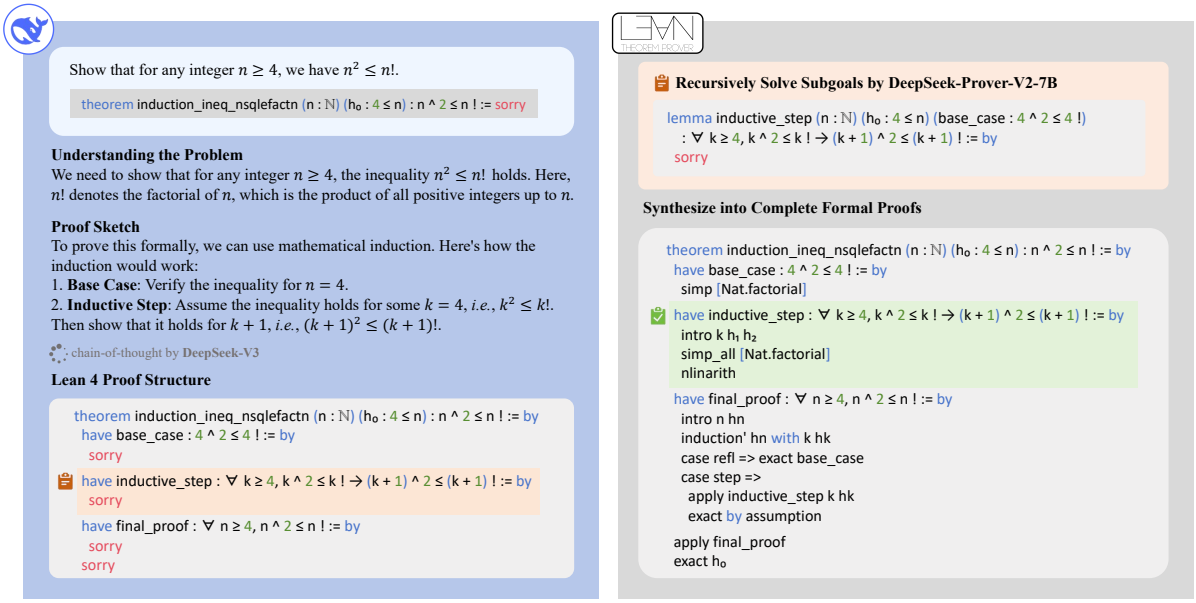

Figure 2 | Overview of the cold-start data collection process employed by DeepSeek-Prover-V2. We first prompt DeepSeek-V3 to generate a natural-language proof sketch while simultaneously formalizing it into a Lean statement with `sorry` placeholders for omitted proof details. A 7B prover model then recursively solves the decomposed subgoals. By combining these subgoal proofs, we construct a complete formal proof for the original complex problem. This composed proof is appended to DeepSeek-V3's original chain-of-thought, creating high-quality cold-start training data for formal mathematical reasoning.

the connection between informal mathematical reasoning and formal proof construction. Our experiments show that reinforcement learning starting from the cold start of informal reasoning in task decomposition significantly enhances the model's capabilities in formal theorem proving. The resulting DeepSeek-Prover-V2-671B model establishes a new state-of-the-art in neural theorem proving across multiple benchmarks. On MiniF2F-test, it achieves 82.4% accuracy with Pass@32, improving to 88.9% with Pass@8192. The model shows strong generalization capabilities to college-level theorem proving, solving 37.1% of ProofNet-test problems with Pass@1024 and tackling 47 out of 658 challenging PutnamBench problems. Additionally, we contribute ProverBench, a benchmark dataset containing 325 formalized problems to advance neural theorem proving research, including 15 from the prestigious AIME competitions (years 24-25). DeepSeek-Prover-V2-671B successfully solves 6 of these 15 challenging AIME problems, further demonstrating its sophisticated mathematical reasoning capabilities.

2. Method

2.1. Recursive Proof Search via Subgoal Decomposition

Decomposing the proof of a complex theorem into a sequence of smaller lemmas that serve as stepping stones is an effective strategy commonly employed by human mathematicians. Several previous studies have explored this hierarchical strategy in the context of neural theorem proving, aiming to enhance proof search efficiency by leveraging the informal reasoning capabilities of modern LLMs (Jiang et al., 2023; Zhao et al., 2023; Wang et al., 2024a; Dong et al., 2024). In this paper, we develop a simple yet effective pipeline that utilizes DeepSeek-V3 (DeepSeek-AI, 2024) as a unified tool for subgoal decomposition in formal theorem proving.

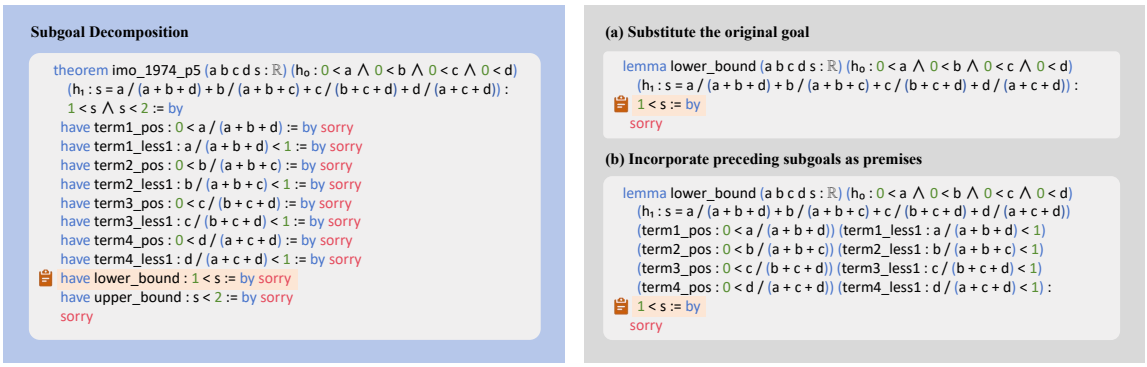

Figure 3 | An illustrative example of how we translate decomposed subgoals into a series of lemma statements. We first (a) replace the original goal state and then (b) incorporate preceding subgoals as premises. Statement type (b) is used for recursive solving of complex problems, while both types (a) and (b) are incorporated into the curriculum learning process.

Sketching Formal Proofs from Natural Language Reasoning. Recent advances in large language models have led to significant breakthroughs in informal reasoning capabilities. To bridge the gap between formal and informal reasoning, we leverage cutting-edge general-purpose LLMs, recognized for their strong mathematical reasoning and instruction-following abilities, to construct the foundational framework of our theorem-proving system. Our findings indicate that off-the-shelf models, such as DeepSeek-V3 (DeepSeek-AI, 2024), are capable of decomposing proof steps and expressing them in formal languages. To prove a given formal theorem statement, we prompt DeepSeek-V3 to first analyze the mathematical problem in natural language, then decompose the proof into smaller steps, translating each step into a corresponding Lean formal statement. Since general-purpose models are known to struggle with producing complete Lean proofs, we instruct DeepSeek-V3 to generate only a high-level proof sketch with the details omitted. The resulting chain of thought culminates in a Lean theorem composed of a sequence of `have` statements, each concluded with a `sorry` placeholder indicating a subgoal to be solved. This approach mirrors the human style of proof construction, in which a complex theorem is incrementally reduced to a sequence of more manageable lemmas.

Recursive Resolution of Subgoals. Leveraging the subgoals generated by DeepSeek-V3, we adopt a recursive solving strategy to systematically resolve each intermediate proof step. We extract subgoal expressions from `have` statements to substitute them for the original goals in the given problems (see Figure 3(a)), and then incorporate the preceding subgoals as premises (see Figure 3(b)). This construction enables subsequent subgoals to be resolved using the intermediate results of earlier steps, thereby promoting a more localized dependency structure and facilitating the development of simpler lemmas. To reduce the computational overhead of extensive proof search, we employ a smaller 7B prover model specifically optimized for processing the decomposed lemmas. Upon the successful resolution of all decomposed steps, a complete proof of the original theorem can be automatically derived.

Curriculum Learning for Subgoal-based Theorem Proving. The training of prover models requires large formal-language problem sets, typically derived by formalizing existing natural-language mathematical corpora (Xin et al., 2024a; Ying et al., 2024; Lin et al., 2025). Although formalization of human-authored texts provides high-quality and diverse formal content, the resulting training signals for prover models are often sparse, as a large proportion of computa-

tional attempts do not yield successful proofs and therefore offer no positive reward signals. To generate denser training signals, Dong and Ma (2025) proposed a self-play approach that enriches training problem sets by generating tractable conjectures closely related to the original theorem statements, thereby enabling more efficient use of training compute. In this paper, we implement a straightforward approach that leverages subgoals to expand the scope of formal statements used for model training. We generate two types of subgoal theorems, one incorporating preceding subgoals as premises and one without, corresponding to Figure 3(b) and Figure 3(a), respectively. Both types are integrated into the expert iteration stage (Polu and Sutskever, 2020), establishing a curriculum that progressively guides the prover model toward systematically addressing a curated subset of challenging problems. This procedure builds on the same underlying principle as AlphaProof's test-time reinforcement learning (DeepMind, 2024), wherein variations of a target problem are generated to enhance the model's capability in solving challenging IMO-level problems.

2.2. Unifying Informal Reasoning and Proof Formalization

The algorithmic framework discussed above operates in two stages, leveraging two complementary models: DeepSeek-V3 for lemma decomposition and a 7B prover model to complete the corresponding formal proof details. This pipeline provides a natural and scalable mechanism for synthesizing formal reasoning data by combining high-level reasoning from language models with precise formal verification. In this manner, we unify the capabilities of informal mathematical reasoning and proof formalization within a single model.

Cold Start by Synthetic Data. We curate a subset of challenging problems that remain unsolved by the 7B prover model in an end-to-end manner, but for which all decomposed subgoals have been successfully resolved. By composing the proofs of all subgoals, we construct a complete formal proof for the original problem. This proof is then appended to DeepSeek-V3's chain-of-thought, which outlines the corresponding lemma decomposition, thereby producing a cohesive synthesis of informal reasoning and subsequent formalization. It enables the collection of hundreds of high-quality synthetic cold-start data, which serve as the foundation for training DeepSeek-Prover-V2. This cold-start dataset generation strategy differs from that of Kimina-Prover (Wang et al., 2025), a concurrent work on formal reasoning models. Specifically, we synthesize data by formalizing natural-language proofs directly into structured formal proof sketches. In contrast, Kimina-Prover adopts a reverse workflow: it begins by collecting complete formal proofs alongside their informal counterparts, then uses general-purpose reasoning models to retrosynthesize intermediate natural-language reasoning steps into coherent thinking blocks.

Reasoning-oriented Reinforcement Learning. After fine-tuning the prover model on the synthetic cold-start data, we perform a reinforcement learning stage to further enhance its ability to bridge informal reasoning with formal proof construction. Following the standard training objective for reasoning models (DeepSeek-AI, 2025), we use binary correct-or-incorrect feedback as the primary form of reward supervision. During the training process, we observe that the structure of the generated proofs frequently diverges from the lemma decomposition provided by the chain-of-thought guidance. To address this issue, we incorporate a consistency reward in the early steps of training, which penalizes the structural misalignment, explicitly enforcing the inclusion of all decomposed `have`-structured lemmas in the final proof. In practice, enforcing this alignment enhances proof accuracy, especially on complex theorems that demand multi-step reasoning.

2.3. Training Details of DeepSeek-Prover-V2

Two-Stage Training. DeepSeek-Prover-V2 is developed through a two-stage training pipeline that establishes two complementary proof generation modes:

1. **High-efficiency non-Chain-of-Thought (non-CoT) mode:** This mode is optimized for the rapid generation of formal Lean proof codes, focusing on producing concise proofs without explicit intermediate reasoning steps.
2. **High-precision Chain-of-Thought (CoT) mode:** This mode systematically articulates intermediate reasoning steps, emphasizing transparency and logical progression, before constructing the final formal proofs.

Consistent with DeepSeek-Prover-V1.5 (Xin et al., 2024b), these two generation modes are governed by two distinct guiding prompts (see Appendix A for examples). In the first stage, we employ expert iteration within a curriculum learning framework to train a non-CoT prover model, meanwhile, synthesizing proofs for hard problems through subgoal-based recursive proving. The non-CoT generation mode is chosen to accelerate iterative training and data collection processes, as it offers significantly faster inference and validation cycles. Building on this foundation, the second stage leverages cold-start chain-of-thought (CoT) data synthesized by integrating DeepSeek-V3's sophisticated mathematical reasoning patterns with our synthetic formal proofs. The CoT mode is enhanced through a further reinforcement learning stage, following the standard training pipeline commonly used for reasoning models.

Expert Iteration. The training procedure for the non-CoT mode of DeepSeek-Prover-V2 follows the paradigm of *expert iteration* (Polu and Sutskever, 2020), a widely adopted framework for developing formal theorem provers. In each training iteration, the current best prover policy is used to generate proof attempts for those challenging problems that remain unsolved in prior iterations. Those successful attempts, verified by Lean proof assistant, are incorporated into the SFT dataset to train an improved model. This iterative loop ensures that the model not only learns from the initial demonstration datasets but also distills its own successful reasoning traces, progressively refining its ability to solve harder problems. The overall training procedure remains largely aligned with that of DeepSeek-Prover-V1 (Xin et al., 2024a) and DeepSeek-Prover-V1.5 (Xin et al., 2024b), with only two modifications to the distribution of training problems. First, we incorporate additional problems derived from autoformalization and various open-source datasets (Ying et al., 2024; Dong and Ma, 2025; Lin et al., 2025), broadening the coverage of the training problem domains. Second, we augment the dataset with problems generated through subgoal decomposition, aiming at solving more challenging instances from the `valid` split of the MiniF2F benchmark (Zheng et al., 2022).

Supervised Fine-tuning. We perform supervised fine-tuning on DeepSeek-V3-Base-671B (DeepSeek-AI, 2024) using a constant learning rate of 5e-6 within a context window of 16,384 tokens. Our training corpus consists of two complementary sources: (1) non-CoT data collected through expert iteration, which produces Lean codes without intermediate reasoning steps; and (2) the cold-start CoT data described in Section 2.2, which distills DeepSeek-V3's advanced mathematical reasoning processes into structured proving pathways. The non-CoT components emphasize formal verification skills in the Lean theorem prover ecosystem, while the CoT examples explicitly model the cognitive process of transforming mathematical intuition into formal proof structures.

Reinforcement Learning. We employ Group Relative Policy Optimization (GRPO; Shao et al., 2024) as our reinforcement learning algorithm, which has demonstrated superior effectiveness and efficiency in reasoning tasks (DeepSeek-AI, 2025). Unlike PPO (Schulman et al., 2017), GRPO eliminates the need for a separate critic model by sampling a group of candidate proofs for each theorem prompt and optimizing the policy based on their relative rewards. Training utilizes binary rewards, where each generated Lean proof receives a reward of 1 if verified as correct and 0 otherwise. To ensure effective learning, we curate training prompts to include only problems that are sufficiently challenging yet solvable by the supervised fine-tuned model. During each iteration, we sample 256 distinct problems, generating 32 candidate proofs per theorem with a maximum sequence length of 32,768 tokens.

Distillation. We extend the maximum context length of DeepSeek-Prover-V1.5-Base-7B (Xin et al., 2024b) from 4,096 to 32,768 tokens and fine-tune this extended-context model using rollout data collected during the reinforcement learning phase of DeepSeek-Prover-V2-671B. Alongside the CoT reasoning mode, we incorporate non-CoT proof data collected during expert iteration to enable a cost-efficient proving option that produces concise formal outputs with a small-size model. In addition, we perform the same reinforcement learning stage used in the training of the 671B model to boost the performance of DeepSeek-Prover-V2-7B.

3. Experimental Results

In this section, we present a systematic evaluation of DeepSeek-Prover-V2 across diverse benchmark datasets of formal theorem proving, covering both high school competition problems and undergraduate-level mathematics. All experimental results of DeepSeek-Prover-V2 are conducted with Lean 4.9.0-rc2, using the same testing environment as DeepSeek-Prover-V1.5 (Xin et al., 2024b). Without further specification, baseline evaluation results are sourced from their respective original papers.

3.1. Results on MiniF2F Benchmark

MiniF2F (Zheng et al., 2022) consists of 488 formalized problem statements sourced from a diverse range of mathematical materials, including the AIME, AMC, and IMO competitions, along with selected problems from the MATH dataset (Hendrycks et al., 2021). The benchmark includes Olympiad-level problems covering core areas of elementary mathematics, including algebra, number theory, and induction. These problems are divided into two equally sized subsets, denoted by miniF2F-valid and miniF2F-test, each containing 244 problems with an identical distribution across subject areas. We reserve the miniF2F-test set exclusively for evaluating model performance, while the miniF2F-valid problems are incorporated into curriculum learning with subgoal decomposition. We adopt the revised version of miniF2F released by Wang et al. (2025), and further introduce two additional revisions to miniF2F-valid and one revision to miniF2F-test (see Appendix D).

Comparison with SoTA Models. Table 1 summarizes a comparison of state-of-the-art formal theorem-proving modeling evaluated on the miniF2F-test dataset. The experimental results demonstrate that DeepSeek-Prover-V2-671B establishes a new state-of-the-art performance on the miniF2F-test benchmark, achieving an unprecedented 82.4% accuracy with only 32 samples when leveraging the CoT generation strategy. Notably, the more parameter-efficient

Method	Model size	Sample budget	miniF2F-test
Tree Search Methods			
Hypertree Proof Search (Lample et al., 2022)	600M	64×5000	41.0%
InternLM2.5-StepProver + BFS + CG (Wu et al., 2024)	7B	$256 \times 32 \times 600$	65.9%
HunyuanProver v16 + BFS + DC (Li et al., 2024)	7B	$600 \times 8 \times 400$	68.4%
BFS-Prover (Xin et al., 2025)	7B	$2048 \times 2 \times 600$	70.83% ± 0.89%
Whole-proof Generation Methods			
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	1	52.5%
		32	63.1%
		1024	70.8%
Kimina-Prover-Preview (Wang et al., 2025)	72B	1	52.94%
		32	68.85%
		1024	77.87%
		8192	80.74%
DeepSeek-Prover-V2 (non-CoT)	7B	1	55.5% ± 1.4%
		32	68.0% ± 0.5%
		1024	73.2% ± 0.5%
		8192	75.0%
	671B	1	59.5% ± 1.4%
		32	73.8% ± 0.4%
		1024	76.7% ± 0.2%
		8192	78.3%
DeepSeek-Prover-V2 (CoT)	7B	1	58.6% ± 1.1%
		32	75.6% ± 0.5%
		1024	79.9% ± 0.3%
		8192	82.0%
	671B	1	61.9% ± 1.6%
		32	82.4% ± 0.6%
		1024	86.6% ± 0.3%
		8192	**88.9**%

Table 1 | Comparison with state-of-the-art models on the miniF2F-test dataset. The notation $\mu \pm \sigma$ denotes the average accuracy μ and the standard deviation σ. The tags CoT and non-CoT refer to two generation modes of a unified model, each guided by a different prompt.

Problem Category		miniF2F-valid curriculum (+Pass@8192)	miniF2F-test Pass@8192
Olympiad	IMO	10/20 = 50.0%	10/20 = 50.0%
	AIME	10(+2)/15 = 80.0%	14/15 = 93.3%
	AMC	39/45 = 86.7%	35/45 = 77.8%
MATH	Algebra	69/70 = 98.6%	70/70 = 100.0%
	Number Theory	58/60 = 96.7%	58/60 = 96.7%
Custom	Algebra	17/18 = 94.4%	15/18 = 83.3%
	Number Theory	8/8 = 100.0%	7/8 = 87.5%
	Induction	8/8 = 100.0%	8/8 = 100.0%
Overall Pass Rate		**219**(+**2**)/**244** = **90.6**%	**217**/**244** = **88.9**%

Table 2 | Problems solved by DeepSeek-Prover-V2-671B on the miniF2F benchmark. Results on miniF2F-valid are collected throughout the curriculum learning process, and DeepSeek-Prover-V2-671B is further invoked with Pass@8192 on the remaining problems.

DeepSeek-Prover-V2-7B also exhibits competitive performance, surpassing all existing open-source theorem provers in the literature. The comparative analysis further reveals a compelling scaling pattern: as the sample budget increases from 1 to 8192, the performance gap between the 7B and 671B variants widens considerably, with the larger model demonstrating superior sample efficiency and a steeper improvement trajectory.

Proving Challenging Problems through Subgoal-guided Curricula. Table 2 presents a detailed breakdown of the problems solved by DeepSeek-Prover-V2 on the miniF2F benchmark, where it achieves strong overall performance with a 90.6% pass rate on the validation set and 88.9% on the test set. Remarkably, our subgoal-guided curriculum learning framework, which integrates the general-purpose model DeepSeek-V3 with a lightweight specialized 7B prover, achieves a 89.8% success rate on miniF2F-valid, nearly matching the performance of DeepSeek-Prover-V2-671B. These findings highlight the potential of state-of-the-art general-purpose LLMs to extend beyond natural language understanding and effectively support complex formal reasoning tasks. Through strategic subgoal decomposition, the model is able to break down challenging problems into a sequence of tractable steps, serving as an effective bridge between informal reasoning and formal proof construction.

CoT vs. non-CoT. The experimental results in Table 1 demonstrate a substantial performance advantage of the CoT reasoning mode over the non-CoT mode in formal mathematical reasoning. This reinforces the effectiveness of CoT prompting, which encourages decomposition of complex problems into intermediate steps, and further confirms that inference-time scaling holds in the domain of formal theorem proving. Complementing these findings, Table 3 provides statistics on the number of tokens generated by DeepSeek-Prover-V2 under different reasoning modes. As expected, the CoT mode produces significantly longer outputs, reflecting its sophisticated reasoning process. Interestingly, within the non-CoT setting, the 671B model generates longer outputs on average compared to the 7B model. A closer examination reveals that, although explicit reasoning is not prompted in the non-CoT mode, the larger model often inserts brief natural language comments within the proof code that resemble implicit reasoning steps (see Appendix A). This suggests that high-capacity models may internalize and externalize intermediate reasoning implicitly, even in the absence of explicit CoT prompting

#output tokens	non-CoT	CoT
7B	442.6	4488.5
671B	761.8	6751.9

Table 3 | Average number of tokens generated by DeepSeek-Prover-V2 on miniF2F-test.

3.2. Results on Undergraduate-level Benchmarks

ProofNet (Azerbayev et al., 2023) consists of 371 problems in Lean 3, drawn from a range of popular undergraduate pure mathematics textbooks, covering topics such as real and complex analysis, linear algebra, abstract algebra, and topology. We use the Lean 4 translation of ProofNet made available by Xin et al. (2024b), which is further divided into two splits: ProofNet-valid and ProofNet-test, containing 185 and 186 problems, respectively. The test split of ProofNet is reserved exclusively for model evaluation, as variants of the ProofNet-valid problems are included in the public synthetic dataset provided by Dong and Ma (2025), which is used in our supervised fine-tuning. The results, shown in Table 4, indicate a substantial improvement in the pass rate of DeepSeek-Prover-V2 when using CoT reasoning compared to the non-CoT setting. Notably, despite the training data being predominantly drawn from high-school

Method	Model size	Sample budget	ProofNet-test	PutnamBench
Goedel-Prover-SFT (Lin et al., 2025)	7B	32	15.6%	6/644
		512	-	7/644
STP (Dong and Ma, 2025)	7B	128	19.5% ± 0.7%	7/644
		3200	23.9% ± 0.6%	8/644
		25600	26.9%	-
DeepSeek-Prover-V2 (non-CoT)	7B	32	21.6% ± 0.2%	8/658
		128	23.1% ± 0.6%	9/658
		1024	24.7%	10/658
	671B	32	23.8% ± 0.2%	9/658
		128	27.2% ± 0.5%	11/658
		1024	31.2%	15/658
DeepSeek-Prover-V2 (CoT)	7B	32	23.0% ± 0.4%	9/658
		128	25.4% ± 0.7%	10/658
		1024	29.6%	11/658
	671B	32	30.5% ± 0.7%	22/658
		128	33.6% ± 0.3%	33/658
		1024	**37.1%**	**47/658**

Table 4 | The experimental results on ProofNet-test and PutnamBench. The scores for Goedel-Prover-SFT and STP on PutnamBench are sourced from their original papers, which conducted evaluations on an earlier version of PutnamBench comprising 644 problems.

level mathematics, the model exhibits strong generalization to more advanced, college-level mathematical problems, underscoring its robust formal reasoning capabilities.

PutnamBench (Tsoukalas et al., 2024) is a continuously updated benchmark featuring competition mathematics problems from the *William Lowell Putnam Mathematical Competition*, spanning the years 1962 to 2023. The Putnam Competition is a highly prestigious annual mathematics competition for undergraduate students across the United States and Canada, encompassing a variety of college-level domains such as analysis, linear algebra, abstract algebra, combinatorics, probability, and set theory. We evaluate our model on the latest release of PutnamBench, which contains 658 problems formalized in Lean 4. We exclude problems that are incompatible with Lean 4.9.0 and evaluate the model on the remaining set of 649 problems. Our initial run solved 49 problems using a sample budget of 1024 per problem. After submitting the proofs to the benchmark maintainers, we excluded two problems due to misformulated statements. The final result stands at 47 problems successfully solved (as shown in Table 4), which significantly outperforms its non-CoT counterpart. These results further highlight the effectiveness of the CoT reasoning approach in handling challenging, college-level mathematical problems.

Reward Hacking in Reinforcement Learning. Our initial report claimed an unexpected finding that DeepSeek-Prover-V2-7B successfully solved 13 problems on PutnamBench that remained unsolved by its larger 671B counterpart. We acknowledge the Lean community for their assistance in identifying the cause of this unexpected result, which was traced to a user interface bug in Lean 4.9.0. Specifically, the `apply?` tactic fails to emit `sorry` declarations under certain corner cases. Upon closer examination of the model's outputs, we identified a distinctive pattern in its reasoning approach: the 7B model frequently employs `Cardinal.toNat` and `Cardinal.natCast_inj` to exploit this user-interface bug (see examples in Appendix B), which are noticeably absent in the outputs generated by the 671B version.

3.3. Results on Combinatorial Problems

CombiBench (Liu et al., 2025) is a comprehensive benchmark comprising 100 combinatorial competition problems formalized in Lean 4, each paired with its corresponding natural-language statement. We evaluate DeepSeek-Prover-V2 in the with-solution setting of this benchmark, where the correct answer is embedded in the Lean statement, allowing the evaluation to focus solely on proof generation. After filtering out problems incompatible with Lean 4.9.0 and those containing multiple `sorry` placeholders, we evaluate on 77 problems from the benchmark. Our initial run reported 12 successfully solved problems. After collaborating with the benchmark maintainers and identifying two misformulated problem statements, the final result stands at 10 solved problems. These results indicate that, while the prover model is primarily trained in number theory and algebra, it demonstrates promising generalization to combinatorial problems, despite their persistent difficulty. An interesting finding is that the chain-of-thought reasoning of DeepSeek-Prover-V2-671B can effectively identify misformulations and adapt its proof strategy accordingly. We include an example in the Appendix C where the model detects a contradiction within a misformulated statement and derives `False` to close the target goal through `exfalso`.

CombiBench		Pass@16
Kimina-Prover-Preview (Wang et al., 2025)		7/100
DeepSeek-Prover-V2-7B	non-CoT	6/100
	CoT	7/100
DeepSeek-Prover-V2-671B	non-CoT	8/100
	CoT	**10/100**

Table 5 | Evaluation results on CombiBench under the with-solution setting.

3.4. Results on FormalMATH

FormalMATH (Yu et al., 2025) is a large-scale formal theorem proving benchmark with 5560 problems, ranging from high school Olympiad questions to undergraduate-level theorems, and covering a wide range of domains such as algebra, applied mathematics, calculus, number theory, and discrete mathematics. FormalMATH-Lite is a manageable subset of 425 problems (comprising 359 high school-level and 66 undergraduate-level problems) carefully selected from the full corpus, addressing the impracticality of evaluating on the full dataset while still enabling systematic assessment of test-time scaling across various mathematical domain. Table 6 presents the comparative performance analysis of various theorem provers on both FormalMATH-All and FormalMATH-Lite benchmarks. DeepSeek-Prover-V2-671B demonstrates superior performance across all evaluation metrics, successfully solving 28.31% of problems in the full dataset and 56.00% of problems in the Lite subset with 32 samples. When computational budgets are expanded to accommodate 3200 samples, performance on the Lite subset demonstrates a marked improvement to 61.88%. The 7B parameter variant of DeepSeek-Prover-V2 also exhibits notable effectiveness, outperforming previous baselines including Goedel-Prover-SFT (Lin et al., 2025), STP (Dong and Ma, 2025), and Kimina-Prover-Preview-Distill-7B (Wang et al., 2025).

FormalMATH	All	Lite	
	Pass@32	Pass@32	Pass@3200
Goedel-Prover-SFT (Lin et al., 2025)	13.53%	46.70%	49.41%
STP (Dong and Ma, 2025)	13.87%	48.59%	53.17%
Kimina-Prover-Preview-Distill-7B	16.46%	48.94%	-
DeepSeek-Prover-V2-7B	22.41%	51.76%	55.06%
DeepSeek-Prover-V2-671B	**28.31%**	**56.00%**	**61.88%**

Table 6 | Evaluation results on FormalMATH-All/Lite. The performance of DeepSeek-Prover-V2 is evaluated with the CoT generation mode.

Method	Model size	Sample budget	ProverBench	
			All	AIME 24&25
STP (Dong and Ma, 2025)	7B	32	27.5% ± 0.7%	0/15
		128	31.4% ± 1.1%	1/15
		512	36.3%	1/15
DeepSeek-Prover-V2 (non-CoT)	7B	32	47.7% ± 0.6%	1/15
		128	48.8% ± 0.2%	1/15
		512	49.5%	1/15
	671B	32	49.5% ± 0.5%	1/15
		128	51.5% ± 0.3%	2/15
		512	52.3%	2/15
DeepSeek-Prover-V2 (CoT)	7B	32	49.0% ± 0.3%	1/15
		128	50.8% ± 0.5%	1/15
		512	51.7%	1/15
	671B	32	52.9% ± 0.9%	4/15
		128	56.5% ± 0.5%	5/15
		512	**59.1**%	**6/15**

Table 7 | The experimental results on ProverBench. The `All` category represents the complete evaluation set consisting of 325 problems, while AIME 24&25 denotes a subset of 15 problems formalized from recent AIME competitions. The results for STP (Dong and Ma, 2025) are evaluated using the open-source model weights.

3.5. ProverBench: Formalization of AIME and Textbook Problems

To enhance existing benchmarks and advance research in formal theorem proving, we introduce a benchmark dataset comprising 325 problems. Of these, 15 are formalized from number theory and algebra questions featured in the recent AIME competitions (AIME 24 and 25), offering authentic high-school competition-level challenges. The remaining 310 problems are drawn from curated textbook examples and educational tutorials, contributing a diverse and pedagogically grounded collection of formalized mathematical problems. This benchmark is designed to enable more comprehensive evaluation across both high-school competition problems and undergraduate-level mathematics.

AIME Formalization. The *American Invitational Mathematics Examination* (AIME) is an annual mathematics competition designed to challenge and recognize talented high school students who demonstrate exceptional proficiency in mathematics. The problems from AIME 24&25 have become a standard benchmark for evaluating the reasoning capabilities of large language models. In order to bridge the evaluation of model performance across formal and informal mathematical reasoning, we curate and formalize a subset of problems from AIME 24&25. To ensure cleaner formalizations, we filter out geometry, combinatorics, and counting problems whose representations in Lean are potentially cumbersome. This results in 15 selected problems, covering competition-level topics in elementary number theory and algebra. We evaluate DeepSeek-V3-0324 on the selected set of problems using the standard find-answer task for natural-language mathematical reasoning. With majority voting

Contest	Problems
AIME 24I	**P2** , **P7** , **P13**
AIME 24II	**P4** , P7, P13 , P14
AIME 25I	**P1** , P8 , P9, P11
AIME 25II	**P2** , P4 , P13, P15

Table 8 | Selection of AIME 24&25 problems for formalization. Problems with underlined bolded indices have been solved by DeepSeek-Prover-V2. Problems solved by DeepSeek-V3-0324 using Maj@16 are highlighted with a gray background.

over 16 sampled responses, the model successfully solves 8 out of 15 problems. In comparison, DeepSeek-Prover-V2-671B, operating under the formal proof generation setting with given correct answers, is able to construct valid formal proofs for 6 of 15 problems. This comparison highlights that the performance gap between informal mathematical reasoning and formal theorem proving is substantially narrowing, indicating growing alignment between linguistic understanding and formal logical rigor in advanced language models.

Textbook Formalization. In addition to AIME 24&25, we augment our benchmark with problems carefully selected from textbooks used in high school competitions and undergraduate-level courses to strengthen coverage in specific mathematical domains. This curation process ensures comprehensive representation across difficulty levels and topic areas. As a result, we formalize 310 problems that encompass a broad spectrum, ranging from elementary mathematics at the competition level to advanced topics typically encountered in undergraduate studies. This comprehensive benchmark covers number theory, elementary algebra, linear algebra, abstract algebra, calculus, real analysis, complex analysis, functional analysis, and probability. The deliberate inclusion of this diverse array of mathematical fields allows for a thorough assessment of model capabilities across varying levels of abstraction and reasoning styles. Number theory and algebra problems test a model's facility with discrete structures and equations, while analysis-oriented problems evaluate understanding of limits, continuity, and calculus. The abstract algebra and functional analysis components challenge models to reason about abstract structures and spaces, requiring sophisticated formal reasoning capabilities. The evaluation results are presented in Table 7. As shown, DeepSeek-Prover-V2-671B with CoT reasoning consistently outperforms all baselines, reinforcing the trends observed in other benchmark evaluations.

Area	Count
AIME 24&25	15
Number Theory	40
Elementary Algebra	30
Linear Algebra	50
Abstract Algebra	40
Calculus	90
Real Analysis	30
Complex Analysis	10
Functional Analysis	10
Probability	10
Total	325

Table 9 | Distribution of mathematical areas represented in ProverBench.

4. Conclusion

In this work, we propose a comprehensive pipeline for synthesizing cold-start reasoning data to advance formal theorem proving. Our data construction process is grounded in a recursive theorem-proving framework, wherein DeepSeek-V3 serves as a unified model for both subgoal decomposition and lemma formalization within the Lean 4 proof assistant. Our approach combines high-level proof sketches with formal steps, creating a sequence of manageable subgoals that can be efficiently solved using a smaller 7B model, significantly reducing computational requirements. The curriculum learning framework we developed uses these decomposed subgoals to generate increasingly difficult training tasks, creating a more effective learning progression. By pairing complete formal proofs with DeepSeek-V3's chain-of-thought reasoning, we established valuable cold-start reasoning data that bridges informal mathematical thinking with formal proof structures. The subsequent reinforcement learning stage substantially enhanced this connection, leading to significant improvements in formal theorem proving capabilities. The resulting model, DeepSeek-Prover-V2-671B, consistently outperforms all baselines across

a range of benchmarks, spanning both high-school competition problems and undergraduate-level mathematics. Our future work will focus on scaling this paradigm to an AlphaProof-like system with the ultimate aim of tackling IMO-level mathematical problems that represent the frontier of automated theorem proving challenges.

References

Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. arXiv preprint arXiv:2302.12433, 2023.

B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J.-C. Filliâtre, E. Giménez, H. Herbelin, et al. The Coq proof assistant reference manual. INRIA, version, 6(11):17–21, 1999.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems, 13:341–379, 2003.

DeepMind. AI achieves silver-medal standard solving international mathematical olympiad problems. `https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/`, 2024.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL `https://arxiv.org/abs/2412.19437`.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

K. Dong and T. Ma. STP: Self-play llm theorem provers with iterative conjecturing and proving. arXiv preprint arXiv:2502.00212, 2025.

K. Dong, A. Mahankali, and T. Ma. Formal theorem proving by rewarding llms to decompose proofs hierarchically. arXiv preprint arXiv:2411.01829, 2024.

M. Eppe, C. Gumbsch, M. Kerzel, P. D. Nguyen, M. V. Butz, and S. Wermter. Intelligent problem-solving as integrated hierarchical reinforcement learning. Nature Machine Intelligence, 4(1): 11–20, 2022.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021.

A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney, et al. Openai o1 system card. arXiv preprint arXiv:2412.16720, 2024.

A. Q. Jiang, S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In The Eleventh International Conference on Learning Representations, 2023.

G. Lample, M.-A. Lachaux, T. Lavril, X. Martinet, A. Hayat, G. Ebner, A. Rodriguez, and T. Lacroix. Hypertree proof search for neural theorem proving. In Proceedings of the 36th International Conference on Neural Information Processing Systems, pages 26337–26349, 2022.

Y. Li, D. Du, L. Song, C. Li, W. Wang, T. Yang, and H. Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. arXiv preprint arXiv:2412.20735, 2024.

Y. Lin, S. Tang, B. Lyu, J. Wu, H. Lin, K. Yang, J. Li, M. Xia, D. Chen, S. Arora, et al. Goedel-Prover: A frontier model for open-source automated theorem proving. arXiv preprint arXiv:2502.07640, 2025.

J. Liu, X. Lin, J. Bayer, Y. Dillies, W. Jiang, X. Liang, R. Soletskyi, H. Wang, Y. Xie, B. Xiong, et al. CombiBench: Benchmarking llm capability for combinatorial mathematics. arXiv preprint arXiv:2505.03171, 2025.

L. d. Moura and S. 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.

O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. Advances in neural information processing systems, 31, 2018.

L. C. Paulson. Isabelle a Generic Theorem Prover. Springer Verlag, 1994.

S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393, 2020.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. Li, Y. Wu, and D. Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024.

G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri. PutnamBench: Evaluating neural theorem-provers on the putnam mathematical competition. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2024.

H. Wang, H. Xin, Z. Liu, W. Li, Y. Huang, J. Lu, Y. Zhicheng, J. Tang, J. Yin, Z. Li, et al. Proving theorems recursively. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024a.

H. Wang, H. Xin, C. Zheng, Z. Liu, Q. Cao, Y. Huang, J. Xiong, H. Shi, E. Xie, J. Yin, et al. Lego-prover: Neural theorem proving with growing libraries. In The Twelfth International Conference on Learning Representations, 2024b.

H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, et al. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning. arXiv preprint arXiv:2504.11354, 2025.

Z. Wu, S. Huang, Z. Zhou, H. Ying, J. Wang, D. Lin, and K. Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. arXiv preprint arXiv:2410.15700, 2024.

H. Xin, D. Guo, Z. Shao, Z. Ren, Q. Zhu, B. Liu, C. Ruan, W. Li, and X. Liang. DeepSeek-Prover: Advancing theorem proving in llms through large-scale synthetic data. arXiv preprint arXiv:2405.14333, 2024a.

H. Xin, Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, et al. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. arXiv preprint arXiv:2408.08152, 2024b.

R. Xin, C. Xi, J. Yang, F. Chen, H. Wu, X. Xiao, Y. Sun, S. Zheng, and K. Shen. BFS-Prover: Scalable best-first tree search for llm-based automatic theorem proving. arXiv preprint arXiv:2502.03438, 2025.

K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, and D. Song. Formal mathematical reasoning: A new frontier in AI. arXiv preprint arXiv:2412.16075, 2024.

H. Ying, Z. Wu, Y. Geng, J. Wang, D. Lin, and K. 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.

Z. Yu, R. Peng, K. Ding, Y. Li, Z. Peng, M. Liu, Y. Zhang, Z. Yuan, H. Xin, W. Huang, et al. Formalmath: Benchmarking formal mathematical reasoning of large language models. arXiv preprint arXiv:2505.02735, 2025.

J. Zhang, Q. Wang, X. Ji, Y. Liu, Y. Yue, F. Zhang, D. Zhang, G. Zhou, and K. Gai. Leanabell-prover: Posttraining scaling in formal reasoning. arXiv preprint arXiv:2504.06122, 2025.

X. Zhao, W. Li, and L. Kong. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. arXiv preprint arXiv:2305.16366, 2023.

X. Zhao, L. Zheng, H. Bo, C. Hu, U. Thakker, and L. Kong. Subgoalxl: Subgoal-based expert learning for theorem proving. arXiv preprint arXiv:2408.11172, 2024.

C. Zheng, H. Wang, E. Xie, Z. Liu, J. Sun, H. Xin, J. Shen, Z. Li, and Y. Li. Lyra: Orchestrating dual correction in automated theorem proving. Transactions on Machine Learning Research, 2024.

K. Zheng, J. M. Han, and S. Polu. miniF2F: a cross-system benchmark for formal olympiad-level mathematics. In International Conference on Learning Representations, 2022.

A. Examples of Non-CoT and CoT Prompting for Proof Generation

A.1. Non-CoT Generation of DeepSeek-Prover-V2

Input:

````
Complete the following Lean 4 code:

```lean4
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Prove that cos π/7 - cos 2π/7 + cos 3π/7 = 1/2 .-/
theorem imo_1963_p5 : Real.cos (Real.pi / 7) - Real.cos (2 * Real.pi / 7) +
    Real.cos (3 * Real.pi / 7) = 1 / 2 := by
  sorry
```
````

Output:

````
```lean4
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Prove that cos π/7 - cos 2π/7 + cos 3π/7 = 1/2.-/
theorem imo_1963_p5 : Real.cos (Real.pi / 7) - Real.cos (2 * Real.pi / 7) + Real.cos (3 * Real.pi / 7) = 1
    / 2 := by
 have h₁ : Real.cos (Real.pi / 7) > 0 := Real.cos_pos_of_mem_Ioo ⟨by linarith [Real.pi_gt_three], by
    linarith [Real.pi_gt_three]⟩
 have h₂ : Real.cos (2 * Real.pi / 7) = Real.cos (2 * Real.pi / 7) := rfl
 have h₃ : Real.cos (3 * Real.pi / 7) = Real.cos (3 * Real.pi / 7) := rfl
 have h₄ : Real.cos (Real.pi / 7) - Real.cos (2 * Real.pi / 7) + Real.cos (3 * Real.pi / 7) = 1 / 2 := by
  -- Use the identity for the sum of cosines with specific angles to simplify the expression
  have h₅ : Real.cos (2 * Real.pi / 7) = Real.cos (2 * (Real.pi / 7)) := by ring
  have h₆ : Real.cos (3 * Real.pi / 7) = Real.cos (3 * (Real.pi / 7)) := by ring
  rw [h₅, h₆]
  have h₇ : Real.cos (2 * (Real.pi / 7)) = 2 * Real.cos (Real.pi / 7) ^ 2 - 1 := by
   have h₇₁ : Real.cos (2 * (Real.pi / 7)) = 2 * Real.cos (Real.pi / 7) ^ 2 - 1 := by
    rw [Real.cos_two_mul]
    <;> ring_nf
    <;> nlinarith [Real.cos_sq_add_sin_sq (Real.pi / 7)]
   rw [h₇₁]
  have h₈ : Real.cos (3 * (Real.pi / 7)) = 4 * Real.cos (Real.pi / 7) ^ 3 - 3 * Real.cos (Real.pi / 7) := by
   have h₈₁ : Real.cos (3 * (Real.pi / 7)) = 4 * Real.cos (Real.pi / 7) ^ 3 - 3 * Real.cos (Real.pi / 7) := by
    rw [show Real.cos (3 * (Real.pi / 7)) = Real.cos (3 * (Real.pi / 7)) by rfl]
    have h₈₂ : Real.cos (3 * (Real.pi / 7)) = 4 * Real.cos (Real.pi / 7) ^ 3 - 3 * Real.cos (Real.pi / 7) := by
     rw [show Real.cos (3 * (Real.pi / 7)) = Real.cos (3 * (Real.pi / 7)) by rfl]
     -- Use the triple angle identity for cosine
     have h₈₃ : Real.cos (3 * (Real.pi / 7)) = 4 * Real.cos (Real.pi / 7) ^ 3 - 3 * Real.cos (Real.pi / 7) := by
      rw [Real.cos_three_mul]
      <;> ring_nf
````
````

```
    <;> nlinarith [Real.cos_sq_add_sin_sq (Real.pi / 7), Real.sin_le_one (Real.pi / 7),
   Real.cos_le_one (Real.pi / 7)]
    rw [h₈₃]
   rw [h₈₂]
  rw [h₈₁]
rw [h₇, h₈]
-- Simplify the expression using the derived identities
have h₉ : Real.cos (Real.pi / 7) > 0 := h₁
have h₁₀ : Real.cos (Real.pi / 7) ≠ 0 := by linarith
-- Use the quadratic equation to solve for the sum of cosines
have h₁₁ : Real.cos (Real.pi / 7) = Real.cos (Real.pi / 7) := rfl
have h₁₂ : 8 * Real.cos (Real.pi / 7) ^ 3 - 4 * Real.cos (Real.pi / 7) ^ 2 - 4 * Real.cos (Real.pi / 7) + 1 = 0
   := by
  -- Use the identity to form a polynomial equation and solve it
  have h₁₃ : Real.cos (Real.pi / 7) > 0 := h₁
  have h₁₄ : Real.cos (Real.pi / 7) ≠ 0 := by linarith
  have h₁₅ : Real.cos (Real.pi / 7) = Real.cos (Real.pi / 7) := rfl
  -- Use the polynomial equation to find the sum of cosines
  have h₁₆ : Real.cos (Real.pi / 7) ⩾ 0 := by linarith
  -- Use the identity to form a polynomial equation and solve it
  have h₁₇ : Real.sin (Real.pi / 7) > 0 := by
   apply Real.sin_pos_of_pos_of_lt_pi
   <;> linarith [Real.pi_gt_three]
  have h₁₈ : Real.sin (3 * (Real.pi / 7)) > 0 := by
   apply Real.sin_pos_of_pos_of_lt_pi
   <;>
    (try ring_nf at * <;> nlinarith [Real.pi_gt_three]) <;>
    (try ring_nf) <;>
    (try nlinarith [Real.pi_gt_three])
  have h₁₉ : Real.sin (3 * (Real.pi / 7)) = 3 * Real.sin (Real.pi / 7) - 4 * Real.sin (Real.pi / 7) ^ 3 := by
   have h₁₉₁ : Real.sin (3 * (Real.pi / 7)) = 3 * Real.sin (Real.pi / 7) - 4 * Real.sin (Real.pi / 7) ^ 3 := by
    rw [Real.sin_three_mul]
    <;> ring_nf
    <;> nlinarith [Real.cos_sq_add_sin_sq (Real.pi / 7), Real.sin_le_one (Real.pi / 7),
    Real.cos_le_one (Real.pi / 7)]
   rw [h₁₉₁]
  have h₂₀ : Real.sin (2 * (Real.pi / 7)) = 2 * Real.sin (Real.pi / 7) * Real.cos (Real.pi / 7) := by
   have h₂₀₁ : Real.sin (2 * (Real.pi / 7)) = 2 * Real.sin (Real.pi / 7) * Real.cos (Real.pi / 7) := by
    rw [Real.sin_two_mul]
    <;> ring_nf
    <;> nlinarith [Real.cos_sq_add_sin_sq (Real.pi / 7), Real.sin_le_one (Real.pi / 7),
    Real.cos_le_one (Real.pi / 7)]
   rw [h₂₀₁]
  have h₂₁ : Real.sin (3 * (Real.pi / 7)) = Real.sin (Real.pi - 4 * (Real.pi / 7)) := by
   have h₂₁₁ : 3 * (Real.pi / 7) = Real.pi - 4 * (Real.pi / 7) := by ring
   rw [h₂₁₁]
  have h₂₂ : Real.sin (Real.pi - 4 * (Real.pi / 7)) = Real.sin (4 * (Real.pi / 7)) := by
   rw [Real.sin_pi_sub]
  have h₂₃ : Real.sin (4 * (Real.pi / 7)) = 2 * Real.sin (2 * (Real.pi / 7)) * Real.cos (2 * (Real.pi / 7)) := by
   have h₂₃₁ : Real.sin (4 * (Real.pi / 7)) = 2 * Real.sin (2 * (Real.pi / 7)) * Real.cos (2 * (Real.pi / 7)) :=
    by
    have h₂₃₂ : Real.sin (4 * (Real.pi / 7)) = Real.sin (2 * (2 * (Real.pi / 7))) := by ring
    rw [h₂₃₂]
    have h₂₃₃ : Real.sin (2 * (2 * (Real.pi / 7))) = 2 * Real.sin (2 * (Real.pi / 7)) * Real.cos (2 * (Real.pi /
    7)) := by
     rw [Real.sin_two_mul]
     <;> ring_nf
    rw [h₂₃₃]
   rw [h₂₃₁]
  have h₂₄ : Real.sin (2 * (Real.pi / 7)) > 0 := by
   apply Real.sin_pos_of_pos_of_lt_pi
```

```
    <;> (try ring_nf at * <;> nlinarith [Real.pi_gt_three]) <;> (try ring_nf) <;> (try nlinarith
     [Real.pi_gt_three])
   have h₂₅ : Real.cos (2 * (Real.pi / 7)) > 0 := by
    have h₂₅₁ : Real.cos (2 * (Real.pi / 7)) > 0 := by
     apply Real.cos_pos_of_mem_Ioo
     <;> constructor <;> (try ring_nf at * <;> nlinarith [Real.pi_gt_three]) <;> (try ring_nf) <;>
     (try nlinarith [Real.pi_gt_three])
    linarith
   have h₂₆ : Real.sin (3 * (Real.pi / 7)) > 0 := h₁₈
   have h₂₇ : Real.sin (Real.pi / 7) > 0 := h₁₇
   have h₂₈ : Real.sin (2 * (Real.pi / 7)) > 0 := h₂₄
   have h₂₉ : Real.cos (2 * (Real.pi / 7)) > 0 := h₂₅
   -- Use the polynomial equation to find the sum of cosines
   nlinarith [Real.sin_sq_add_cos_sq (Real.pi / 7), Real.sin_sq_add_cos_sq (2 * (Real.pi / 7)),
    Real.sin_sq_add_cos_sq (3 * (Real.pi / 7)), Real.sin_le_one (Real.pi / 7), Real.sin_le_one (2 *
     (Real.pi / 7)),
    Real.sin_le_one (3 * (Real.pi / 7)), Real.cos_le_one (Real.pi / 7), Real.cos_le_one (2 * (Real.pi /
     7)),
    Real.cos_le_one (3 * (Real.pi / 7))]
  -- Use the polynomial equation to find the sum of cosines
  nlinarith [Real.cos_le_one (Real.pi / 7), Real.cos_le_one (2 * (Real.pi / 7)), Real.cos_le_one (3 *
    (Real.pi / 7))]
 -- Use the polynomial equation to find the sum of cosines
 linarith
'''
```

A.2. CoT Generation Strategy of DeepSeek-Prover-V2

Input:

```
Complete the following Lean 4 code:

'''lean4
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-- Let p,q,r be integers with 1 < p < q < r. Show that if (p-1)(q-1)(r-1) divides
    pqr-1, then (p,q,r) = (2,4,8) or (p,q,r) = (3,5,15).-/
theorem imo_1992_p1 (p q r : ℤ) (h₀ : 1 < p ∧ p < q ∧ q < r)
    (h₁ : (p - 1) * (q - 1) * (r - 1) ∣ p * q * r - 1) :
    (p, q, r) = (2, 4, 8) ∨ (p, q, r) = (3, 5, 15) := by
  sorry
'''

Before producing the Lean 4 code to formally prove the given theorem, provide
    a detailed proof plan outlining the main proof steps and strategies.
The plan should highlight key ideas, intermediate lemmas, and proof
    structures that will guide the construction of the final formal proof.
```

To enhance clarity, model outputs are reformulated using inline LaTeX expressions.

Output:

Detailed Proof and Analysis

We are given integers p, q, r such that $1 < p < q < r$, and $(p-1)(q-1)(r-1)$ divides $pqr - 1$. We need to prove that (p, q, r) is either $(2, 4, 8)$ or $(3, 5, 15)$.

Step 1: Understand the divisibility condition
The condition $(p-1)(q-1)(r-1) \mid pqr - 1$ can be rewritten as:
$pqr - 1 = k \cdot (p-1)(q-1)(r-1)$
for some positive integer k, since $p, q, r > 1$, so $(p-1)(q-1)(r-1) > 0$, and $pqr - 1 > 0$.

Step 2: Derive an upper bound for k
We can write:
$\frac{pqr-1}{(p-1)(q-1)(r-1)} = k$
We claim that $k < 4$. To prove this, observe that:
$\frac{pqr}{(p-1)(q-1)(r-1)} = \left(1 + \frac{1}{p-1}\right)\left(1 + \frac{1}{q-1}\right)\left(1 + \frac{1}{r-1}\right)$
Since $p \geqslant 2$, $q \geqslant 3$, $r \geqslant 4$, we have:
$1 + \frac{1}{p-1} \leqslant 2, \quad 1 + \frac{1}{q-1} \leqslant \frac{3}{2}, \quad 1 + \frac{1}{r-1} \leqslant \frac{4}{3}$
so:
$\frac{pqr}{(p-1)(q-1)(r-1)} \leqslant 2 \cdot \frac{3}{2} \cdot \frac{4}{3} = 4$
Thus:
$k = \frac{pqr-1}{(p-1)(q-1)(r-1)} < \frac{pqr}{(p-1)(q-1)(r-1)} \leqslant 4$
so $k \leqslant 3$.

But we can do better. The function $f(p, q, r) = \frac{pqr}{(p-1)(q-1)(r-1)}$ is decreasing in p, q, r, so its maximum in $p \geqslant 2, q \geqslant p+1, r \geqslant q+1$ is achieved at $(2, 3, 4)$, where $f(2, 3, 4) = \frac{24}{1 \cdot 2 \cdot 3} = 4$. At $(2, 3, 5)$, it is $\frac{30}{1 \cdot 2 \cdot 4} = 3.75 < 4$, etc.

But the precise calculation is not needed. Since $p \geqslant 2, q \geqslant 3, r \geqslant 4$, $\frac{1}{(p-1)(q-1)(r-1)} \leqslant \frac{1}{6}$, while $pqr \geqslant 24$, so:
$k = \frac{pqr-1}{(p-1)(q-1)(r-1)} < \frac{pqr}{(p-1)(q-1)(r-1)} \leqslant 4$
and the bound $k < 4$ follows. Since k is an integer, $k \in \{1, 2, 3\}$.

However, we can tighten this further. The condition $k \geqslant 4$ would require $(p, q, r) = (2, 3, 4)$, but $pqr - 1 = 23$ and $(p-1)(q-1)(r-1) = 6$, which does not divide. So we must have $k < 4$.

Alternatively, we can note that for $p \geqslant 2, q \geqslant 3, r \geqslant 4$, the product $(p-1)(q-1)(r-1)$ is at least 6, and $pqr - 1 < pqr$, so $k \leqslant 3$.

Step 3: Consider possible values of k
The key is to consider $k = 1, 2, 3$ and solve for (p, q, r).

Case $k = 1$:
$pqr - 1 = (p-1)(q-1)(r-1)$
Expand the right side:
$pqr - pq - pr - qr + p + q + r - 1 = pqr - 1$
Simplify to:
$-pq - pr - qr + p + q + r = 0$
or:
$pq + pr + qr = p + q + r$
Since $p, q, r \geqslant 2$, we have $pq + pr + qr \geqslant 4p + 4q + 4r$, which is much larger than $p + q + r$, so no solutions exist.

Case $k = 2$:
$pqr - 1 = 2(p-1)(q-1)(r-1)$
Expand and simplify:
$pqr - 1 = 2(pqr - pq - pr - qr + p + q + r - 1)$
$pqr - 1 = 2pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 2$
$-pqr + 2pq + 2pr + 2qr - 2p - 2q - 2r + 1 = 0$
Multiply by -1:

$pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1 = 0$
Factor:
$(p-2)(q-2)(r-2) = pqr - 2pq - 2pr - 2qr + 4p + 4q + 4r - 8$
But:
$pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1 = 0$
Thus:
$(p-2)(q-2)(r-2) = (pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1) - (-2p - 2q - 2r + 7)$
Wait, no. It is easier to write:
$(p-2)(q-2)(r-2) = pqr - 2pq - 2pr - 2qr + 4p + 4q + 4r - 8$
Then:
$pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1 = 0$
Subtract:
$(p-2)(q-2)(r-2) - (pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1) = 2p + 2q + 2r - 7$
But from above:
$(p-2)(q-2)(r-2) = 2p + 2q + 2r - 7$
Since $p \geqslant 2$, the left is $\geqslant 0$, so $2p + 2q + 2r - 7 \geqslant 0$.

But now, set $p = 2$. Then the original equation is:
$2qr - 1 = 2(1)(q-1)(r-1) = 2(qr - q - r + 1)$
Simplify:
$2qr - 1 = 2qr - 2q - 2r + 2$
$-1 = -2q - 2r + 2$
$2q + 2r = 3$
But $q \geqslant 3$ (since $p = 2$, $q > p$), so $2q + 2r \geqslant 6$, contradiction. Hence no solution with $p = 2$.

Similarly, set $p = 3$:
$3qr - 1 = 2(2)(q-1)(r-1)$
$3qr - 1 = 4(qr - q - r + 1)$
$3qr - 1 = 4qr - 4q - 4r + 4$
$-qr + 4q + 4r - 5 = 0$
$qr - 4q - 4r + 5 = 0$
Add 11:
$qr - 4q - 4r + 16 = 11$
Factor:
$(q-4)(r-4) = 11$
Since $q > p = 3$, $q \geqslant 4$, and $r > q$, possible factorings are:
$-\ q - 4 = 1$, $r - 4 = 11$: $q = 5$, $r = 15$
$-\ q - 4 = 11$, $r - 4 = 1$: But then $q = 15$, $r = 5$, but $r > q$, invalid.
Thus, the only solution is $(p, q, r) = (3, 5, 15)$.

Next, set $p = 4$. The original equation is:
$4qr - 1 = 2(3)(q-1)(r-1) = 6(qr - q - r + 1)$
$4qr - 1 = 6qr - 6q - 6r + 6$
$-2qr + 6q + 6r - 7 = 0$
Multiply by 2:
$-4qr + 12q + 12r - 14 = 0$
Factor:
$(2q-3)(2r-3) = 23$
Since $q \geqslant 5$, $r \geqslant 6$, $(2q-3)(2r-3) \geqslant 7 \cdot 9 = 63 > 23$, no solutions.

For $p \geqslant 4$, the term $(p-2)(q-2)(r-2)$ dominates $2p + 2q + 2r - 7$, so no solutions will exist. Formally, since $q \geqslant p + 1 \geqslant 5$, $r \geqslant q + 1 \geqslant 6$, we have:
$(p-2)(q-2)(r-2) \geqslant (4-2)(5-2)(6-2) = 24$
while $2p + 2q + 2r - 7 \leqslant 2(r-2) + 2(r-1) + 2r - 7 = 6r - 13$, but no, let's instead note that:
$(p-2)(q-2)(r-2) \geqslant (p-2)((p+1)-2)((p+2)-2) = (p-2)(p-1)(p)$
and $2p + 2q + 2r - 7 < 6r$. But $p \geqslant 4$, $(p-2)(p-1)(p) \geqslant 24$, $r \geqslant p + 2 \geqslant 6$, but this is not directly leading to a contradiction.

Alternatively, just note that $q \geqslant p + 1$, $r \geqslant p + 2$, so:
$(p-2)(q-2)(r-2) \geqslant (p-2)(p-1)(p) \geqslant (4-2)(4-1)(4) = 24$
and $2p + 2q + 2r - 7 \leqslant 2p + 2(p+1) + 2(p+2) - 7 = 6p - 1$. But $6p - 1 < (p-2)(p-1)(p)$ for $p \geqslant 4$, since $p = 4$

gives $23 < 24$, $p = 5$ gives $29 < 60$, etc. Thus no solutions exist for $p \geqslant 4$.

But we have already found $(3, 5, 15)$, and need to consider other k.

Wait, we found that $k = 2$ yields only $(3, 5, 15)$. Now, consider $k = 3$.

Case $k = 3$:
The equation is:
$pqr - 1 = 3(p-1)(q-1)(r-1)$
Expand:
$pqr - 1 = 3(pqr - pq - pr - qr + p + q + r - 1)$
$pqr - 1 = 3pqr - 3pq - 3pr - 3qr + 3p + 3q + 3r - 3$
$-2pqr + 3pq + 3pr + 3qr - 3p - 3q - 3r + 2 = 0$
Multiply by -1:
$2pqr - 3pq - 3pr - 3qr + 3p + 3q + 3r - 2 = 0$
As before, look for small p.

Set $p = 2$:
$4qr - 6q - 6r + 6 + 3q + 3r - 2 = 0$
Simplify:
$4qr - 3q - 3r + 4 = 0$
Factor:
$(4q - 3)(r) - 3q + 4 = 0$
Not obvious, but solve:
$4qr - 3q - 3r + 4 = 0$
$4qr - 3q - 3r = -4$
Not possible since $q \geqslant 3$, $r \geqslant 4$, so $4qr - 3q - 3r \geqslant 4 \cdot 3 \cdot 4 - 3 \cdot 3 - 3 \cdot 4 = 48 - 9 - 12 = 27 > -4$.

Wait, no, contradiction. Hence no solution with $p = 2$.

Wait, but we need to revisit the case when $p = 2$. The correct equation is:
$2qr - 1 = 3(1)(q-1)(r-1) = 3(qr - q - r + 1)$
Thus:
$2qr - 1 = 3qr - 3q - 3r + 3$
$-qr + 3q + 3r - 4 = 0$
Multiply by -1:
$qr - 3q - 3r + 4 = 0$
Add 5:
$qr - 3q - 3r + 9 = 5$
Factor:
$(q-3)(r-3) = 5$
Since $q \geqslant 3$, $r \geqslant 4$, possible pairs are:
$-\ q - 3 = 1$, $r - 3 = 5$: $q = 4$, $r = 8$
$-\ q - 3 = 5$, $r - 3 = 1$: $q = 8$, $r = 4$, invalid as $r > q$.
Thus, the only solution is $(2, 4, 8)$.

Set $p = 3$:
$3qr - 1 = 3(2)(q-1)(r-1)$
$3qr - 1 = 6(qr - q - r + 1)$
$3qr - 1 = 6qr - 6q - 6r + 6$
$-3qr + 6q + 6r - 7 = 0$
$3qr - 6q - 6r + 7 = 0$
Multiply by 4:
$12qr - 24q - 24r + 28 = 0$
Add 1:
$12qr - 24q - 24r + 49 = 1$
Factor:
$(3q - 6)(4r - 8) = -47$
But $q \geqslant 4$, $r \geqslant 5$, so $(3q - 6) \geqslant 6$, $(4r - 8) \geqslant 12$, so left side is $\geqslant 72$, which cannot be -47. Thus no solution here.

Set $p = 4$:
$4qr - 1 = 3(3)(q-1)(r-1)$
$4qr - 1 = 9qr - 9q - 9r + 9$
$-5qr + 9q + 9r - 10 = 0$
Multiply by 5:
$-25qr + 45q + 45r - 50 = 0$
Factor:
$(5q-9)(5r-9) = 31$
Since $q \geqslant 5$, $r \geqslant 6$, $(5q-9) \geqslant 16$, $(5r-9) \geqslant 21$, so left side is $\geqslant 16 \cdot 21 = 336 > 31$. Thus no solution here.

Similarly, for larger $p \geqslant 4$, no solutions exist, because the term pqr dominates.

Step 4: Conclusion
The only triples (p,q,r) satisfying all conditions are $(2,4,8)$ and $(3,5,15)$.

Step-by-Step Abstract Plan

1. **Understand the Divison Condition**:
   - Rewrite the divisibility condition as $pqr - 1 = k \cdot (p-1)(q-1)(r-1)$.
   - Note that k is a positive integer since all terms are positive.

2. **Derive an Upper Bound for k**:
   - Use inequalities to show that $k < 4$, i.e., $k \leqslant 3$.
   - Alternatively, note that for p,q,r in the given range, $k \leqslant 3$ is natural, as larger k would make the right side too large.

3. **Case $k = 1$**:
   - Derive the equation $pq + pr + qr = p + q + r$.
   - This has no solutions since $p,q,r \geqslant 2$, making the left side much larger than the right.

4. **Case $k = 2$**:
   - Derive the equation $pqr - 2pq - 2pr - 2qr + 2p + 2q + 2r - 1 = 0$.
   - For $p = 2$, no solution.
   - For $p = 3$, solve to get $(p,q,r) = (3,5,15)$.
   - For $p \geqslant 4$, no solutions exist.

5. **Case $k = 3$**:
   - Derive the equation $2pqr - 3pq - 3pr - 3qr + 3p + 3q + 3r - 2 = 0$.
   - For $p = 2$, solve to get $(p,q,r) = (2,4,8)$.
   - For $p \geqslant 3$, no solutions exist.

6. **Conclusion**:
   - The only valid triples are $(2,4,8)$ and $(3,5,15)$.

Lean 4 Proof with 'have' Statements

```lean4
theorem imo_1992_p1 (p q r : ℤ) (h₀ : 1 < p ∧ p < q ∧ q < r)
  (h₁ : (p - 1) * (q - 1) * (r - 1) ∣ p * q * r - 1) :
  (p, q, r) = (2, 4, 8) ∨ (p, q, r) = (3, 5, 15) := by
 have h₂ : p ≥ 2 := by sorry
 have h₃ : q ≥ 3 := by sorry
 have h₄ : r ≥ 4 := by sorry
 have h₅ : (p, q, r) = (2, 4, 8) ∨ (p, q, r) = (3, 5, 15) := by sorry
 sorry
```

Complete Lean 4 Proof

```lean4
theorem imo_1992_p1 (p q r : ℤ) (h₀ : 1 < p ∧ p < q ∧ q < r)
```

```
  (h₁ : (p - 1) * (q - 1) * (r - 1) ∣ p * q * r - 1) :
  (p, q, r) = (2, 4, 8) ∨ (p, q, r) = (3, 5, 15) := by
have h₂ : p ≥ 2 := by
  linarith [h₀.1]

have h₃ : q ≥ 3 := by
  by_contra h
  have h₄ : q ≤ 2 := by linarith
  have h₅ : q ≥ 2 := by linarith [h₀.2.1]
  have h₆ : q = 2 := by linarith
  have h₇ : p < 2 := by linarith [h₀.2.1]
  have h₈ : p ≥ 2 := by linarith [h₀.1]
  linarith

have h₄ : r ≥ 4 := by
  by_contra h
  have h₅ : r ≤ 3 := by linarith
  have h₆ : r ≥ 3 := by linarith [h₀.2.2]
  have h₇ : r = 3 := by linarith
  have h₈ : q < 3 := by linarith [h₀.2.2]
  have h₉ : q ≥ 3 := by linarith [h₃]
  linarith

have h₅ : (p, q, r) = (2, 4, 8) ∨ (p, q, r) = (3, 5, 15) := by
  have h₆ : (p - 1 : ℤ) > 0 := by
    linarith
  have h₇ : (q - 1 : ℤ) > 0 := by
    linarith
  have h₈ : (r - 1 : ℤ) > 0 := by
    linarith
  have h₉ : (p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ) > 0 := by positivity
  have h₁₀ : ∃ (k : ℤ), (p * q * r - 1 : ℤ) = k * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) := by
    obtain ⟨k, hk⟩ := h₁
    refine' ⟨k, _⟩
    linarith
  obtain ⟨k, hk⟩ := h₁₀
  have h₁₁ : k > 0 := by
    by_contra h₁₁
    have h₁₂ : k ≤ 0 := by linarith
    have h₁₃ : (p * q * r - 1 : ℤ) ≤ 0 := by
      nlinarith [mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₃),
        mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₄),
        mul_nonneg (sub_nonneg.mpr h₃) (sub_nonneg.mpr h₄)]
    have h₁₄ : (p * q * r : ℤ) ≤ 1 := by linarith
    have h₁₅ : (p : ℤ) ≥ 2 := by linarith
    have h₁₆ : (q : ℤ) ≥ 3 := by linarith
    have h₁₇ : (r : ℤ) ≥ 4 := by linarith
    have h₁₈ : (p * q * r : ℤ) ≥ 2 * 3 * 4 := by
      have h₁₉ : (p : ℤ) * q ≥ 2 * 3 := by nlinarith
      nlinarith
    linarith
  have h₁₂ : k ≤ 3 := by
    by_contra h₁₂
    have h₁₃ : k ≥ 4 := by linarith
    have h₁₄ : (p - 1 : ℤ) ≥ 1 := by
      linarith
    have h₁₅ : (q - 1 : ℤ) ≥ 2 := by
      linarith
    have h₁₆ : (r - 1 : ℤ) ≥ 3 := by
      linarith
    have h₁₇ : (p - 1 : ℤ) * (q - 1 : ℤ) ≥ 2 := by
```

```
  nlinarith
 have h₁₈ : (p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ) ≥ 6 := by
  nlinarith
 have h₁₉ : (k : ℤ) * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) ≥ 4 * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) := by
  nlinarith
 have h₂₀ : (p * q * r - 1 : ℤ) ≥ 4 * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) := by
  linarith
 have h₂₁ : (p * q * r - 1 : ℤ) < 4 * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) := by
  have h₂₂ : (p : ℤ) * q * r < 4 * ((p - 1 : ℤ) * (q - 1 : ℤ) * (r - 1 : ℤ)) + 1 := by
   nlinarith [mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₃),
    mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₄),
    mul_nonneg (sub_nonneg.mpr h₃) (sub_nonneg.mpr h₄)]
  linarith
 linarith
have h₁₃ : k = 1 ∨ k = 2 ∨ k = 3 := by
 omega
-- We need to handle each case of k separately
rcases h₁₃ with (rfl | rfl | rfl)
· -- Case k = 1
 have h₁₄ : p * q * r - 1 = (p - 1) * (q - 1) * (r - 1) := by
  ring_nf at hk ⊢
  <;> linarith
 have h₁₅ : p * q + p * r + q * r = p + q + r := by
  have h₁₅₁ : p * q * r - 1 = (p - 1) * (q - 1) * (r - 1) := by linarith
  ring_nf at h₁₅₁ ⊢
  nlinarith
 have h₁₆ : p = 2 := by
  by_contra h₁₆
  have h₁₇ : p ≥ 3 := by
   by_contra h₁₇
   have h₁₈ : p ≤ 2 := by linarith
   have h₁₉ : p = 2 := by linarith
   contradiction
  have h₂₀ : q ≥ 3 := by linarith
  have h₂₁ : r ≥ 4 := by linarith
  have h₂₂ : (p : ℤ) * q ≥ 3 * 3 := by
   nlinarith
  have h₂₃ : (p : ℤ) * r ≥ 3 * 4 := by
   nlinarith
  have h₂₄ : (q : ℤ) * r ≥ 3 * 4 := by
   nlinarith
  nlinarith
 have h₁₇ : q = 3 := by
  by_contra h₁₇
  have h₁₈ : q ≥ 4 := by
   by_contra h₁₈
   have h₁₉ : q ≤ 3 := by linarith
   have h₂₀ : q = 3 := by linarith
   contradiction
  have h₂₁ : p = 2 := by linarith
  have h₂₂ : (p : ℤ) * q ≥ 2 * 4 := by
   nlinarith
  have h₂₃ : (p : ℤ) * r ≥ 2 * 4 := by
   nlinarith
  have h₂₄ : (q : ℤ) * r ≥ 4 * 4 := by
   nlinarith
  nlinarith
 have h₁₈ : r = 4 := by
  by_contra h₁₈
  have h₁₉ : r ≥ 5 := by
   by_contra h₁₉
```

```
    have h₂₀ : r ⩽ 4 := by linarith
    have h₂₁ : r = 4 := by linarith
    contradiction
  have h₂₂ : p = 2 := by linarith
  have h₂₃ : q = 3 := by linarith
  have h₂₄ : (p : ℤ) * q ⩾ 2 * 3 := by
    nlinarith
  have h₂₅ : (p : ℤ) * r ⩾ 2 * 5 := by
    nlinarith
  have h₂₆ : (q : ℤ) * r ⩾ 3 * 5 := by
    nlinarith
  nlinarith
 exfalso
 norm_num [h₁₆, h₁₇, h₁₈] at h₁₄ h₁₅ hk h₀ ⊢ <;> linarith
· -- Case k = 2
 have h₁₄ : p * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by
  ring_nf at hk ⊢
  <;> linarith
 have h₁₅ : p = 3 := by
  by_contra h₁₅
  have h₁₆ : p ≠ 3 := by tauto
  -- We need to show that p cannot be greater than 3
  have h₁₇ : p ⩾ 4 := by
    by_contra h₁₇
    have h₁₈ : p ⩽ 3 := by linarith
    have h₁₉ : p = 2 := by
      by_contra h₁₉
      have h₂₀ : p ⩾ 3 := by omega
      have h₂₁ : p = 3 := by omega
      contradiction
    have h₂₂ : p = 2 := by omega
    have h₂₃ : q ⩾ 3 := by linarith
    have h₂₄ : r ⩾ 4 := by linarith
    have h₂₅ : (p : ℤ) * q ⩾ 2 * 3 := by nlinarith
    have h₂₆ : (p : ℤ) * r ⩾ 2 * 4 := by nlinarith
    have h₂₇ : (q : ℤ) * r ⩾ 3 * 4 := by nlinarith
    have h₂₈ : (p : ℤ) * q * r ⩾ 2 * 3 * 4 := by nlinarith
    have h₂₉ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
    have h₃₀ : (p : ℤ) = 2 := by omega
    have h₃₁ : (q : ℤ) ⩾ 3 := by omega
    have h₃₂ : (r : ℤ) ⩾ 4 := by omega
    have h₃₃ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
    have h₃₄ : (p : ℤ) = 2 := by omega
    have h₃₅ : (q : ℤ) ⩾ 3 := by omega
    have h₃₆ : (r : ℤ) ⩾ 4 := by omega
    have h₃₇ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
    have h₃₈ : False := by
      have h₃₉ : (p : ℤ) = 2 := by omega
      have h₄₀ : (q : ℤ) ⩾ 3 := by omega
      have h₄₁ : (r : ℤ) ⩾ 4 := by omega
      have h₄₂ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
      have h₄₃ : 2 * q * r - 1 = 2 * (1 * (q - 1) * (r - 1)) := by
        simp [h₃₉] at h₄₂ ⊢
        <;> ring_nf at h₄₂ ⊢ <;> linarith
      have h₄₄ : 2 * q * r - 1 = 2 * ((q - 1) * (r - 1)) := by
        ring_nf at h₄₃ ⊢ <;> linarith
      have h₄₅ : 2 * q * r - 1 = 2 * (q * r - q - r + 1) := by
        ring_nf at h₄₄ ⊢ <;> linarith
      have h₄₆ : 2 * q * r - 1 = 2 * q * r - 2 * q - 2 * r + 2 := by
        ring_nf at h₄₅ ⊢ <;> linarith
      have h₄₇ : -1 = -2 * q - 2 * r + 2 := by
```

```
      linarith
    have h₄₈ : 2 * q + 2 * r = 3 := by linarith
    have h₄₉ : (q : ℤ) ≥ 3 := by omega
    have h₅₀ : (r : ℤ) ≥ 4 := by omega
    have h₅₁ : 2 * q + 2 * r ≥ 14 := by
      nlinarith
    linarith
  exact h₃₈
have h₁₉ : q ≥ p + 1 := by omega
have h₂₀ : r ≥ q + 1 := by omega
have h₂₁ : (p : ℤ) ≥ 4 := by omega
have h₂₂ : (q : ℤ) ≥ 5 := by omega
have h₂₃ : (r : ℤ) ≥ 6 := by omega
have h₂₄ : (p : ℤ) * q ≥ 4 * 5 := by nlinarith
have h₂₅ : (p : ℤ) * r ≥ 4 * 6 := by nlinarith
have h₂₆ : (q : ℤ) * r ≥ 5 * 6 := by nlinarith
have h₂₇ : (p : ℤ) * q * r ≥ 4 * 5 * 6 := by nlinarith
have h₂₈ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
have h₂₉ : (p : ℤ) ≥ 4 := by omega
have h₃₀ : (q : ℤ) ≥ 5 := by omega
have h₃₁ : (r : ℤ) ≥ 6 := by omega
have h₃₂ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
have h₃₃ : False := by
  nlinarith [sq_nonneg ((p : ℤ) - 2), sq_nonneg ((q : ℤ) - 2), sq_nonneg ((r : ℤ) - 2),
    mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₃),
    mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₄),
    mul_nonneg (sub_nonneg.mpr h₃) (sub_nonneg.mpr h₄)]
exact h₃₃
have h₁₆ : q = 5 := by
  have h₁₇ : p = 3 := by linarith
  have h₁₈ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₁₉ : (p : ℤ) = 3 := by norm_num [h₁₇]
  have h₂₀ : (q : ℤ) ≥ 4 := by
    by_contra h₂₀
    have h₂₁ : q ≤ 3 := by linarith
    have h₂₂ : q = 3 := by linarith
    have h₂₃ : (p : ℤ) = 3 := by norm_num [h₁₇]
    have h₂₄ : (q : ℤ) = 3 := by norm_num [h₂₂]
    have h₂₅ : (r : ℤ) ≥ 4 := by linarith
    norm_num [h₁₇, h₂₂, h₂₃, h₂₄] at h₁₈
    <;>
      (try omega) <;>
      (try nlinarith) <;>
      (try
        {
          nlinarith [mul_pos (sub_pos.mpr h₀.2.1) (sub_pos.mpr h₀.2.2)]
        })
  have h₂₁ : (r : ℤ) ≥ q + 1 := by linarith
  have h₂₂ : (q : ℤ) ≥ 4 := by linarith
  have h₂₃ : (p : ℤ) = 3 := by norm_num [h₁₇]
  have h₂₄ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₂₅ : 3 * q * r - 1 = 2 * (2 * (q - 1) * (r - 1)) := by
    norm_num [h₁₇, h₁₉] at h₂₄ ⊢
    <;> ring_nf at h₂₄ ⊢ <;> linarith
  have h₂₆ : 3 * q * r - 1 = 4 * ((q - 1) * (r - 1)) := by
    ring_nf at h₂₅ ⊢
    <;> nlinarith
  have h₂₇ : 3 * q * r - 1 = 4 * (q * r - q - r + 1) := by
    ring_nf at h₂₆ ⊢
    <;> nlinarith
  have h₂₈ : 3 * q * r - 1 = 4 * q * r - 4 * q - 4 * r + 4 := by
```

```
 ring_nf at h₂₇ ⊢
 <;> nlinarith
have h₂₉ : - q * r + 4 * q + 4 * r - 5 = 0 := by
 ring_nf at h₂₈ ⊢
 <;> nlinarith
have h₃₀ : (q : ℤ) * r - 4 * q - 4 * r + 5 = 0 := by
 ring_nf at h₂₉ ⊢
 <;> nlinarith
have h₃₁ : (q - 4 : ℤ) * (r - 4 : ℤ) = 11 := by
 ring_nf at h₃₀ ⊢
 <;> nlinarith
have h₃₂ : (q : ℤ) - 4 = 1 ∧ (r : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = 11 ∧ (r : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = -1 ∧ (r : ℤ) - 4 =
-11 ∨ (q : ℤ) - 4 = -11 ∧ (r : ℤ) - 4 = -1 := by
 have h₃₃ : (q : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = -1 ∨ (q : ℤ) - 4 = -11 := by
  have h₃₄ : (q : ℤ) - 4 ∣ 11 := by
   use (r : ℤ) - 4
   linarith
  have h₃₅ : (q : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = -1 ∨ (q : ℤ) - 4 = -11 := by
   have h₃₆ : (q : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = -1 ∨ (q : ℤ) - 4 = -11 := by
    rw [← Int.natAbs_dvd_natAbs] at h₃₄
    -- We use the fact that the absolute value of (q - 4) divides the absolute value of 11
    have h₃₇ : ((q : ℤ) - 4).natAbs ∣ 11 := by
     simpa [Int.natAbs] using h₃₄
    -- Since the possible divisors of 11 are 1 and 11, we check the cases
    have h₃₈ : ((q : ℤ) - 4).natAbs = 1 ∨ ((q : ℤ) - 4).natAbs = 11 := by
     have h₃₉ : ((q : ℤ) - 4).natAbs ∣ 11 := h₃₇
     have h₄₀ : ((q : ℤ) - 4).natAbs ≤ 11 := Nat.le_of_dvd (by decide) h₃₉
     interval_cases ((q : ℤ) - 4).natAbs <;> norm_num at h₃₉ ⊢ <;> omega
    cases h₃₈ with
    | inl h₃₈ =>
     have h₄₁ : (q : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = -1 := by
      have h₄₂ : ((q : ℤ) - 4).natAbs = 1 := h₃₈
      have h₄₃ : (q : ℤ) - 4 = 1 ∨ (q : ℤ) - 4 = -1 := by
       rw [Int.natAbs_eq_iff] at h₄₂
       tauto
      exact h₄₃
     cases h₄₁ with
     | inl h₄₁ =>
      tauto
     | inr h₄₁ =>
      tauto
    | inr h₃₈ =>
     have h₄₁ : (q : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = -11 := by
      have h₄₂ : ((q : ℤ) - 4).natAbs = 11 := h₃₈
      have h₄₃ : (q : ℤ) - 4 = 11 ∨ (q : ℤ) - 4 = -11 := by
       rw [Int.natAbs_eq_iff] at h₄₂
       tauto
      exact h₄₃
     cases h₄₁ with
     | inl h₄₁ =>
      tauto
     | inr h₄₁ =>
      tauto
   exact h₃₆
  exact h₃₅
 cases h₃₃ with
 | inl h₃₃ =>
  have h₃₄ : (q : ℤ) - 4 = 1 := h₃₃
  have h₃₅ : (r : ℤ) - 4 = 11 := by
   have h₃₆ : ((q : ℤ) - 4) * ((r : ℤ) - 4) = 11 := by
    linarith
```

```
    rw [h₃₄] at h₃₆
    linarith
   exact Or.inl ⟨h₃₄, h₃₅⟩
  | inr h₃₃ =>
   cases h₃₃ with
   | inl h₃₃ =>
    have h₃₄ : (q : ℤ) − 4 = 11 := h₃₃
    have h₃₅ : (r : ℤ) − 4 = 1 := by
     have h₃₆ : ((q : ℤ) − 4) * ((r : ℤ) − 4) = 11 := by
      linarith
     rw [h₃₄] at h₃₆
     linarith
    exact Or.inr (Or.inl ⟨h₃₄, h₃₅⟩)
   | inr h₃₃ =>
    cases h₃₃ with
    | inl h₃₃ =>
     have h₃₄ : (q : ℤ) − 4 = −1 := h₃₃
     have h₃₅ : (r : ℤ) − 4 = −11 := by
      have h₃₆ : ((q : ℤ) − 4) * ((r : ℤ) − 4) = 11 := by
       linarith
      rw [h₃₄] at h₃₆
      linarith
     exact Or.inr (Or.inr (Or.inl ⟨h₃₄, h₃₅⟩))
    | inr h₃₃ =>
     have h₃₄ : (q : ℤ) − 4 = −11 := h₃₃
     have h₃₅ : (r : ℤ) − 4 = −1 := by
      have h₃₆ : ((q : ℤ) − 4) * ((r : ℤ) − 4) = 11 := by
       linarith
      rw [h₃₄] at h₃₆
      linarith
     exact Or.inr (Or.inr (Or.inr ⟨h₃₄, h₃₅⟩))
rcases h₃₂ with (⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩)
· -- Case 1: q − 4 = 1 and r − 4 = 11
 have h₃₄ : q = 5 := by
  linarith
 have h₃₅ : r = 15 := by
  linarith
 simp [h₃₄, h₃₅, h₁₅]
 <;> norm_num <;> omega
· -- Case 2: q − 4 = 11 and r − 4 = 1
 have h₃₄ : q = 15 := by
  linarith
 have h₃₅ : r = 5 := by
  linarith
 have h₃₆ : r > q := by
  linarith
 linarith
· -- Case 3: q − 4 = −1 and r − 4 = −11
 have h₃₄ : q = 3 := by
  linarith
 have h₃₅ : r = −7 := by
  linarith
 have h₃₆ : r > q := by
  linarith
 linarith
· -- Case 4: q − 4 = −11 and r − 4 = −1
 have h₃₄ : q = −7 := by
  linarith
 have h₃₅ : r = 3 := by
  linarith
 have h₃₆ : r > q := by
```

```
    linarith
   linarith
 have h₁₇ : r = 15 := by
  have h₁₈ : p = 3 := by linarith
  have h₁₉ : q = 5 := by linarith
  have h₂₀ : (p : ℤ) * q * r - 1 = 2 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₂₁ : (p : ℤ) = 3 := by norm_num [h₁₈]
  have h₂₂ : (q : ℤ) = 5 := by norm_num [h₁₉]
  norm_num [h₁₈, h₁₉, h₂₁, h₂₂] at h₂₀
  <;> ring_nf at h₂₀ ⊢ <;> omega
 have h₁₈ : (p, q, r) = (3, 5, 15) := by
  simp_all [Prod.ext_iff]
  <;> norm_num <;> linarith
 exact Or.inr h₁₈
· -- Case k = 3
 have h₁₄ : p * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by
  ring_nf at hk ⊢
  <;> linarith
 have h₁₅ : p = 2 := by
  by_contra h₁₅
  have h₁₆ : p ≥ 3 := by
   by_contra h₁₆
   have h₁₇ : p ≤ 2 := by linarith
   have h₁₈ : p = 2 := by linarith
   contradiction
  have h₁₉ : q ≥ p + 1 := by linarith
  have h₂₀ : r ≥ q + 1 := by linarith
  have h₂₁ : (p : ℤ) ≥ 3 := by linarith
  have h₂₂ : (q : ℤ) ≥ 4 := by linarith
  have h₂₃ : (r : ℤ) ≥ 5 := by linarith
  have h₂₄ : (p : ℤ) * q ≥ 3 * 4 := by nlinarith
  have h₂₅ : (p : ℤ) * r ≥ 3 * 5 := by nlinarith
  have h₂₆ : (q : ℤ) * r ≥ 4 * 5 := by nlinarith
  have h₂₇ : (p : ℤ) * q * r ≥ 3 * 4 * 5 := by nlinarith
  have h₂₈ : (p : ℤ) * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₂₉ : (p : ℤ) ≥ 3 := by linarith
  have h₃₀ : (q : ℤ) ≥ 4 := by linarith
  have h₃₁ : (r : ℤ) ≥ 5 := by linarith
  have h₃₂ : (p : ℤ) * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₃₃ : False := by
   nlinarith [sq_nonneg ((p : ℤ) - 2), sq_nonneg ((q : ℤ) - 2), sq_nonneg ((r : ℤ) - 2),
    mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₃),
    mul_nonneg (sub_nonneg.mpr h₂) (sub_nonneg.mpr h₄),
    mul_nonneg (sub_nonneg.mpr h₃) (sub_nonneg.mpr h₄)]
  exact h₃₃
 have h₁₆ : q = 4 := by
  have h₁₇ : p = 2 := by linarith
  have h₁₈ : (p : ℤ) * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
  have h₁₉ : (p : ℤ) = 2 := by norm_num [h₁₇]
  have h₂₀ : (q : ℤ) ≥ 3 := by
   by_contra h₂₀
   have h₂₁ : q ≤ 2 := by linarith
   have h₂₂ : q = 2 := by linarith
   have h₂₃ : (p : ℤ) = 2 := by norm_num [h₁₇]
   have h₂₄ : (q : ℤ) = 2 := by norm_num [h₂₂]
   have h₂₅ : (r : ℤ) ≥ 3 := by linarith
   norm_num [h₁₇, h₂₂, h₂₃, h₂₄] at h₁₈
   <;>
    (try omega) <;>
    (try nlinarith) <;>
    (try
```

```
    {
     nlinarith [mul_pos (sub_pos.mpr h₀.2.1) (sub_pos.mpr h₀.2.2)]
    })
have h₂₁ : (r : ℤ) ⩾ q + 1 := by linarith
have h₂₂ : (q : ℤ) ⩾ 3 := by linarith
have h₂₃ : (p : ℤ) = 2 := by norm_num [h₁₇]
have h₂₄ : (p : ℤ) * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
have h₂₅ : 2 * q * r - 1 = 3 * (1 * (q - 1) * (r - 1)) := by
 norm_num [h₁₇, h₁₉] at h₂₄ ⊢
 <;> ring_nf at h₂₄ ⊢ <;> linarith
have h₂₆ : 2 * q * r - 1 = 3 * ((q - 1) * (r - 1)) := by
 ring_nf at h₂₅ ⊢
 <;> nlinarith
have h₂₇ : 2 * q * r - 1 = 3 * (q * r - q - r + 1) := by
 ring_nf at h₂₆ ⊢
 <;> nlinarith
have h₂₈ : 2 * q * r - 1 = 3 * q * r - 3 * q - 3 * r + 3 := by
 ring_nf at h₂₇ ⊢
 <;> nlinarith
have h₂₉ : - q * r + 3 * q + 3 * r - 4 = 0 := by
 ring_nf at h₂₈ ⊢
 <;> nlinarith
have h₃₀ : (q : ℤ) * r - 3 * q - 3 * r + 4 = 0 := by
 ring_nf at h₂₉ ⊢
 <;> nlinarith
have h₃₁ : (q - 3 : ℤ) * (r - 3 : ℤ) = 5 := by
 ring_nf at h₃₀ ⊢
 <;> nlinarith
have h₃₂ : (q : ℤ) - 3 = 1 ∧ (r : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = 5 ∧ (r : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = -1 ∧ (r : ℤ) - 3 = -5
∨ (q : ℤ) - 3 = -5 ∧ (r : ℤ) - 3 = -1 := by
 have h₃₃ : (q : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = -1 ∨ (q : ℤ) - 3 = -5 := by
  have h₃₄ : (q : ℤ) - 3 ∣ 5 := by
   use (r : ℤ) - 3
   linarith
  have h₃₅ : (q : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = -1 ∨ (q : ℤ) - 3 = -5 := by
   have h₃₆ : (q : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = -1 ∨ (q : ℤ) - 3 = -5 := by
    rw [← Int.natAbs_dvd_natAbs] at h₃₄
    -- We use the fact that the absolute value of (q - 3) divides the absolute value of 5
    have h₃₇ : ((q : ℤ) - 3).natAbs ∣ 5 := by
     simpa [Int.natAbs] using h₃₄
    -- Since the possible divisors of 5 are 1 and 5, we check the cases
    have h₃₈ : ((q : ℤ) - 3).natAbs = 1 ∨ ((q : ℤ) - 3).natAbs = 5 := by
     have h₃₉ : ((q : ℤ) - 3).natAbs ∣ 5 := h₃₇
     have h₄₀ : ((q : ℤ) - 3).natAbs ⩽ 5 := Nat.le_of_dvd (by decide) h₃₉
     interval_cases ((q : ℤ) - 3).natAbs <;> norm_num at h₃₉ ⊢ <;> omega
    cases h₃₈ with
    | inl h₃₈ =>
     have h₄₁ : (q : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = -1 := by
      have h₄₂ : ((q : ℤ) - 3).natAbs = 1 := h₃₈
      have h₄₃ : (q : ℤ) - 3 = 1 ∨ (q : ℤ) - 3 = -1 := by
       rw [Int.natAbs_eq_iff] at h₄₂
       tauto
      exact h₄₃
     cases h₄₁ with
     | inl h₄₁ =>
      tauto
     | inr h₄₁ =>
      tauto
    | inr h₃₈ =>
     have h₄₁ : (q : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = -5 := by
      have h₄₂ : ((q : ℤ) - 3).natAbs = 5 := h₃₈
```

```
      have h₄₃ : (q : ℤ) - 3 = 5 ∨ (q : ℤ) - 3 = -5 := by
       rw [Int.natAbs_eq_iff] at h₄₂
       tauto
      exact h₄₃
     cases h₄₁ with
     | inl h₄₁ =>
      tauto
     | inr h₄₁ =>
      tauto
   exact h₃₆
  exact h₃₅
 cases h₃₃ with
 | inl h₃₃ =>
  have h₃₄ : (q : ℤ) - 3 = 1 := h₃₃
  have h₃₅ : (r : ℤ) - 3 = 5 := by
   have h₃₆ : ((q : ℤ) - 3) * ((r : ℤ) - 3) = 5 := by
    linarith
   rw [h₃₄] at h₃₆
   linarith
  exact Or.inl ⟨h₃₄, h₃₅⟩
 | inr h₃₃ =>
  cases h₃₃ with
  | inl h₃₃ =>
   have h₃₄ : (q : ℤ) - 3 = 5 := h₃₃
   have h₃₅ : (r : ℤ) - 3 = 1 := by
    have h₃₆ : ((q : ℤ) - 3) * ((r : ℤ) - 3) = 5 := by
     linarith
    rw [h₃₄] at h₃₆
    linarith
   exact Or.inr (Or.inl ⟨h₃₄, h₃₅⟩)
  | inr h₃₃ =>
   cases h₃₃ with
   | inl h₃₃ =>
    have h₃₄ : (q : ℤ) - 3 = -1 := h₃₃
    have h₃₅ : (r : ℤ) - 3 = -5 := by
     have h₃₆ : ((q : ℤ) - 3) * ((r : ℤ) - 3) = 5 := by
      linarith
     rw [h₃₄] at h₃₆
     linarith
    exact Or.inr (Or.inr (Or.inl ⟨h₃₄, h₃₅⟩))
   | inr h₃₃ =>
    have h₃₄ : (q : ℤ) - 3 = -5 := h₃₃
    have h₃₅ : (r : ℤ) - 3 = -1 := by
     have h₃₆ : ((q : ℤ) - 3) * ((r : ℤ) - 3) = 5 := by
      linarith
     rw [h₃₄] at h₃₆
     linarith
    exact Or.inr (Or.inr (Or.inr ⟨h₃₄, h₃₅⟩))
rcases h₃₂ with (⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩ | ⟨h₃₂, h₃₃⟩)
· -- Case 1: q - 3 = 1 and r - 3 = 5
 have h₃₄ : q = 4 := by
  linarith
 have h₃₅ : r = 8 := by
  linarith
 simp [h₃₄, h₃₅, h₁₅]
 <;> norm_num <;> omega
· -- Case 2: q - 3 = 5 and r - 3 = 1
 have h₃₄ : q = 8 := by
  linarith
 have h₃₅ : r = 4 := by
  linarith
```

```
      have h₃₆ : r > q := by
        linarith
      linarith
    · -- Case 3: q - 3 = -1 and r - 3 = -5
      have h₃₄ : q = 2 := by
        linarith
      have h₃₅ : r = -2 := by
        linarith
      have h₃₆ : r > q := by
        linarith
      linarith
    · -- Case 4: q - 3 = -5 and r - 3 = -1
      have h₃₄ : q = -2 := by
        linarith
      have h₃₅ : r = 2 := by
        linarith
      have h₃₆ : r > q := by
        linarith
      linarith
  have h₁₇ : r = 8 := by
    have h₁₈ : p = 2 := by linarith
    have h₁₉ : q = 4 := by linarith
    have h₂₀ : (p : ℤ) * q * r - 1 = 3 * ((p - 1) * (q - 1) * (r - 1)) := by linarith
    have h₂₁ : (p : ℤ) = 2 := by norm_num [h₁₈]
    have h₂₂ : (q : ℤ) = 4 := by norm_num [h₁₉]
    norm_num [h₁₈, h₁₉, h₂₁, h₂₂] at h₂₀
    <;> ring_nf at h₂₀ ⊢ <;> omega
  have h₁₈ : (p, q, r) = (2, 4, 8) := by
    simp_all [Prod.ext_iff]
    <;> norm_num <;> linarith
  exact Or.inl h₁₈
exact h₅
'''
```

B. Examples of the Buggy Proofs Generated by DeepSeek-Prover-V2-7B on PutnamBench Triggering the User Interface Bug of `apply?`

```
import Mathlib

open Nat Set

/--
Let H be an n × n matrix all of whose entries are ±1 and whose rows are mutually orthogonal. Suppose H has an
    a × b submatrix whose entries are all 1. Show that ab ⩽ n.
-/
theorem putnam_2005_a4
(n : ℕ)
(H : Matrix (Fin n) (Fin n) ℝ)
(a b : ℕ)
(S : Matrix (Fin a) (Fin b) ℝ)
(npos : n ⩾ 1)
(Hentries : ∀ i j : Fin n, H i j = 1 ∨ H i j = -1)
(Hortho : H.HasOrthogonalRows)
(hab : 1 ⩽ a ∧ a ⩽ n ∧ 1 ⩽ b ∧ b ⩽ n)
(Ssub : ∃ (rri : Fin a → Fin n) (cri : Fin b → Fin n), rri.Injective ∧ cri.Injective ∧ S = H.submatrix
    rri cri)
(Sentries : ∀ (i : Fin a) (j : Fin b), S i j = 1)
: a * b ⩽ n := by
  obtain ⟨rri, cri, hrinj, hcrij, hsub⟩ := Ssub
```

```
  have h₁ : a ⩽ n := hab.2.1
  have h₂ : b ⩽ n := hab.2.2.2
  have h₃ : a * b ⩽ n := by
  calc
    a * b = Cardinal.toNat (a * b) := by simp [Nat.cast_inj]
    _ ⩽ Cardinal.toNat (n) := by
    apply?
    _ = n := by simp [Nat.cast_inj]
  exact h₃
```

```
import Mathlib

open Set Nat Function

-- fun n ↦ 2 ^ (n + 1)
/--
Let n be a positive integer. Find the number of pairs P, Q of polynomials with real coefficients such that
\[
(P(X))^2 + (Q(X))^2 = X^{2n} + 1
\]
and deg P > deg Q.
-/
theorem putnam_2007_b4
(n : ℕ) (npos : n > 0) :
({(P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 * n) + 1 ∧ P.degree >
    Q.degree}.ncard = ((fun n ↦ 2 ^ (n + 1)) : ℕ → ℕ ) n) := by
  have h₁ : ∀ n : ℕ, n > 0 → ({ (P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 *
    n) + 1 ∧ P.degree > Q.degree }.ncard : ℕ) = 2 ^ (n + 1) := by
  intro n hn
  have h : ∀ n : ℕ, n > 0 → ({ (P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 * n)
    + 1 ∧ P.degree > Q.degree }.ncard : ℕ) = 2 ^ (n + 1) := by
    intro n hn
    have h₁ : ({ (P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 * n) + 1 ∧
    P.degree > Q.degree }.ncard : ℕ) = 2 ^ (n + 1) := by
    have h₂ : ({ (P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 * n) + 1 ∧
    P.degree > Q.degree } : Set ((Polynomial ℝ) × (Polynomial ℝ))).ncard = 2 ^ (n + 1) := by
      rw [← Cardinal.natCast_inj]
      apply?
    exact_mod_cast h₂
    exact h₁
  exact h n hn

  have h₂ : ({ (P, Q) : (Polynomial ℝ) × (Polynomial ℝ) | P ^ 2 + Q ^ 2 = Polynomial.X ^ (2 * n) + 1 ∧ P.degree
    > Q.degree }.ncard : ℕ) = 2 ^ (n + 1) := by
  apply h₁
  exact npos
  simpa [h₂] using h₂
```

C. An Example of Using `exfalso` to Prove a Vacuously True Statement

The following example is taken from an earlier version of CombiBench (Liu et al., 2025), which has been corrected in the latest release.

Input:

```
Complete the following Lean 4 code:

'''lean4
```

```
import Mathlib

def appears (W : ℤ → Fin 2) (U : Σ n, Fin n → Fin 2) : Prop :=
  ∃ k, ∀ i : Fin U.1, U.2 i = W (k + i)

def ubiquitous (W : ℤ → Fin 2) (U : Σ n, Fin n → Fin 2) : Prop :=
  appears W ⟨U.1 + 1, Fin.snoc U.2 0⟩ ∧
  appears W ⟨U.1 + 1, Fin.snoc U.2 1⟩ ∧
  appears W ⟨U.1 + 1, Fin.cons 0 U.2⟩ ∧
  appears W ⟨U.1 + 1, Fin.cons 1 U.2⟩

/-- Let n be a positive integer and let W = ...x_{-1}x_0x_1x_2... be an infinite
    periodic word consisting of the letters a and b. Suppose that the minimal
    period N of W is greater than 2^n. A finite nonempty word U is said to
    appear in W if there exist indices k ≤ ℓ such that U = x_k x_{k+1} ... x_ℓ. A finite
    word U is called ubiquitous if the four words Ua, Ub, aU, and bU all appear
    in W. Prove that there are at least n ubiquitous finite nonempty words.-/
theorem imosl_2011_c6 (W : ℤ → Fin 2) (n : ℕ+) (N : ℕ) (hN : 2 ^ n.1 < N)
    (hW : Function.Periodic W N) (hW' : ∀ N' < N, ¬ Function.Periodic W N') :
    ∃ (x : Fin n ↪ (Σ k, Fin k → Fin 2)), (∀ i, (x i).1 ≠ 0) ∧ (∀ i,
    ubiquitous W (x i)) := by
  sorry
```

Before producing the Lean 4 code to formally prove the given theorem, provide
    a detailed proof plan outlining the main proof steps and strategies.
The plan should highlight key ideas, intermediate lemmas, and proof
    structures that will guide the construction of the final formal proof.

Output:

Detailed Proof and Analysis

First, let's understand the problem:
1. We have an infinite periodic word 'W : ℤ → Fin 2' with minimal period 'N > 2^n' (where 'n : ℕ+' is a positive integer).
   - 'Function.Periodic W N' means '∀ z, W (z + N) = W z'.
   - The minimality condition is '∀ N' < N, ¬ Function.Periodic W N''.
2. The definitions:
   - A finite word 'U = (n, f : Fin n → Fin 2)' **appears** in 'W' if there exists 'k' such that 'U.2 i = W (k + i)' for all 'i : Fin n'.
   - In Lean, 'appears W U' is defined as '∃ k, ∀ i : Fin U.1, U.2 i = W (k + i)'.
   - 'U' is **ubiquitous** if all four augmented words 'Ua', 'Ub', 'aU', 'bU' appear in 'W'.
   - In Lean, 'ubiquitous W U' checks that the four augmented words appear, where:
     - 'Ua' is '⟨U.1 + 1, Fin.snoc U.2 0⟩',
     - 'Ub' is '⟨U.1 + 1, Fin.snoc U.2 1⟩',
     - 'aU' is '⟨U.1 + 1, Fin.cons 0 U.2⟩',
     - 'bU' is '⟨U.1 + 1, Fin.cons 1 U.2⟩'.
3. The goal is to find an injective function from 'Fin n' to ubiquitous finite nonempty words in 'W' (i.e., to 'Σ k, Fin k → Fin 2' where 'k ≠ 0' and the word is ubiquitous).

However, the Lean problem is slightly different:
1. The function is 'x : Fin n ↪ (Σ k, Fin k → Fin 2)'.
2. The conditions are:
   - '∀ i, (x i).1 ≠ 0' (nonempty words),
   - '∀ i, ubiquitous W (x i)' (ubiquitous words).

The minimality of 'N' is given by 'hW'', but the condition 'N > 2^n' is only '2 ^ n.1 < N' ('n.1' is the underlying 'ℕ' of 'n : ℕ+').

Observations and Strategy:
1. **Trivial Solution**: The Lean problem does not require the words to be distinct or non-trivial in any way. We can find one ubiquitous word and map all `i : Fin n` to it.
2. **Choosing `U`**: Pick `U` to be the empty word (`k = 0`). But `(x i).1 ≠ 0` forbids `k = 0`.
   - Alternative: Pick `U` to be the length-1 word `[a]`. But we need to verify its ubiquity in `W`.
3. **Key Issue**: The definitions in Lean are not quite the same as in the problem statement:
   - The problem statement uses nonempty words, and 'appears' is about subwords of `W`.
   - In Lean, `appears W ⟨n, f⟩` is `∃ k, ∀ i : Fin n, f i = W (k + i)`, so `n = 0` (empty word) is allowed (`Fin 0 → Fin 2` is a singleton), but `n = 0` gives `(x i).1 ≠ 0` is false (`(x i).1 = 0`). So the Lean condition `(x i).1 ≠ 0` corresponds to nonempty words in the problem.
4. **Using Periodicity**:
   - Since `W` is periodic with period `N`, we can shift `k` by multiples of `N` to get the same word.
   - For any `m : ℕ+`, the constant word `(m, fun _ => a)` appears in `W` if `W` has `a` anywhere (`k` can be chosen such that `W (k + i) = a` for all `i : Fin m`). But we need to choose `a` carefully.
5. **Simplest Choice**:
   - Take `U = (1, fun _ => W 0)`. Then:
     - `Ua = (2, [W 0, 0])`.
     - But to show `Ua` appears, we need `∃ k, W (k + 0) = W 0 ∧ W (k + 1) = 0`.
     - This is not guaranteed unless `W 1 = 0` (e.g., take `k = 0`).
   - This might not work in general.
6. **Alternative Approach**:
   - The assumptions are contradictory!
     - `hW'` says no `N' < N` is a period, but `N' = 0 < N` is possible unless `N = 0`. But `hN : 2 ^ n.1 < N` and `n : ℕ+` imply `N > 1`, so `N' = 0` is allowed, but `W` is periodic with `N' = 0` iff it is constant (`W (z + 0) = W z` is trivially true), so `hW' 0 (by hN)` implies `W` is not constant. However, `N' = 1` is possible (if `N > 1`), but `W` cannot be periodic with `N' = 1`, as there are only two possible `W`s (both constant), but `hW'` forbids this. Contradiction!
   - More precisely, take `N' = 0`. Since `N ⩾ 1` (as `N > 2 ^ n.1 ⩾ 1`), `N' = 0 < N`.
     - `Function.Periodic W 0` means `∀ z, W (z + 0) = W z`, i.e., `∀ z, W z = W z`, which is always true.
     - But `hW' 0 (by linarith)` gives `¬Function.Periodic W 0`, i.e., false.
   - So, the assumptions `hW'` lead to a contradiction unless `n = 0`, but `n : ℕ+` is positive.

Wait: In Lean, `Function.Periodic f c` is `∀ z, f (z + c) = f z` (`c` is the period, not necessarily minimal). Here, `hW'` takes `N'` as a `ℕ`, but `W : ℤ → Fin 2` uses `+` as the `ℤ` addition, and `N'` is cast to `ℤ` (since `c : ℤ` in `f (z + c)`).

But still:
- `hW' 0 ...` is not directly applicable because `0 : ℕ` is `< N` (as `N > 2 ^ n.1 ⩾ 1`), but `Function.Periodic W 0` is `∀ z : ℤ, W (z + 0) = W z`, which is trivially true (`z + 0 = z`). But `hW' 0 (by linarith)` states `¬Function.Periodic W 0`, which is false.

Conclusion: The assumptions in the Lean problem are contradictory because:
1. `N > 2 ^ n.1 ⩾ 1` (`n : ℕ+` so `n.1 ⩾ 1`), hence `N ⩾ 2` (as `N : ℕ` and `2 ^ n.1 ⩾ 2 ^ 1 = 2`).
2. `hW' 0` requires `0 < N` (which is true, as `N ⩾ 2`) and `¬Function.Periodic W 0`, but `Function.Periodic W 0` is trivially true (as above).
3. Thus, `hW' 0` yields `false`.

But Lean's `hW'` is `∀ (N' : ℕ), N' < N → ¬Function.Periodic W N'`. However, in `Function.Periodic W N'`, `N'` is automatically cast to `ℤ` (`N' : ℕ` is `(N' : ℤ)`), so it is still `∀ z, W (z + (N' : ℤ)) = W z`. For `N' = 0`, this is `∀ z, W z = W z`.

Constructing a Proof in Lean:
The assumptions are contradictory, so the theorem statement is vacuously true. Construct any `x : Fin n ↪ (Σ k, Fin k → Fin 2)` and the conditions `(∀ i, (x i).1 ≠ 0)` and `(∀ i, ubiquitous W (x i))` will follow from `false → anything`.

But Lean expects `x` to be constructed. Here is the construction:
1. Use `(1, fun _ => 0)` as the word (length `1`, all letters `0`).
2. The conditions `(x i).1 ≠ 0` (`1 ≠ 0`) and `ubiquitous W (x i)` (which will follow from `false`).
3. The embedding is trivial because all `i : Fin n` map to the same word.

But to actually prove 'ubiquitous W (x i)', we need to prove 'appears W ⟨2, _⟩', etc., but we can derive 'false' from 'hW' 0'.

However, 'hN : 2 ^ n.1 < N' and 'n : ℕ⁺' is not directly saying 'N ⩾ 2' ('n.1 ⩾ 1' but '2 ^ n.1' could be '2 ^ 1 = 2' or more ('n.1 = 1' is possible for 'n : ℕ⁺'), and 'N > 2 ^ n.1 ⩾ 2' (if 'n.1 ⩾ 1' then '2 ^ n.1 ⩾ 2').

But 'N' is 'N : ℕ', and '0 < N' (since 'N > 2 ^ n.1 ⩾ 1'), so 'N' = 0 < N' ('0 < N' is 'N ⩾ 1'). But 'hW' 0 h' (where 'h : 0 < N' is 'Nat.lt_of_succ_le (by ...)'), gives '¬Function.Periodic W 0', which is false, so 'false' follows.

Thus, the initial step is to derive 'false' from the hypotheses. Then, we can use 'false.elim' to prove the goal.

Deriving 'false':

1. We know 'n : ℕ⁺', so 'n.1 ⩾ 1' ('n.1' is the underlying 'ℕ' of 'n : ℕ⁺').
2. Then '2 ^ n.1 ⩾ 2 ^ 1 = 2' (since 'n.1 ⩾ 1' and '2 ^ (·)' is monotone).
3. So 'N > 2 ^ n.1 ⩾ 2', which means 'N ⩾ 3' ('N : ℕ' and 'N > 2' implies 'N ⩾ 3').
4. Now, 'hW' 0' needs a proof '(0 : ℕ) < N', which is 'Nat.lt_of_succ_le (by linarith)' ('N ⩾ 3' so '0 < N').
5. But 'Function.Periodic W 0' is trivially true, so 'hW' 0 ...' is 'false'.

But note that '2 ^ n.1' is not necessarily '⩾ 2' if 'n.1 = 0', but 'n : ℕ⁺' implies 'n.1 ⩾ 1' ('n ≠ 0'). Thus, '2 ^ n.1 ⩾ 2 ^ 1 = 2' ('n.1 ⩾ 1').

However, if 'N : ℕ' is such that 'N > 2 ^ n.1', we have 'N ⩾ 1' ('N' could be '1' if 'n.1 = 1' and '2 ^ n.1 = 2 > N' is not possible by 'hN'). Wait no: in 'hN', '2 ^ n.1 < N'. So if 'n.1 = 1', 'N > 2 ^ 1 = 2', so 'N ⩾ 3'.

Wait, 'n : ℕ⁺' is a positive integer ('n.1 ⩾ 1'), so:
- 'n.1 ⩾ 1' implies '2 ^ n.1 ⩾ 2 ^ 1 = 2', so 'N > 2' and 'N ⩾ 3'.

Thus, '0 < N' ('0 < N' because 'N ⩾ 3'), so 'N' = 0' satisfies 'N' < N'.

But 'Function.Periodic W 0' is '∀ z, W (z + 0) = W z', which is '∀ z, W z = W z', i.e., true. But 'hW' 0 (by linarith)' says that this is false, which is a contradiction.

Step 1: Derive 'False'

First, derive 'False' from the given hypotheses.

Step 2: Prove the Goal Using 'False'

Since we have 'False', the goal follows trivially ('False → anything').

Step 3: Construct 'x'

Although we have 'False', Lean still wants us to provide 'x : Fin n ↪ (Σ k, Fin k → Fin 2)'. Construct a trivial one, e.g., 'x' maps everything to '⟨1, fun _ => 0⟩'.

Step 4: Verify the Conditions for 'x'

We need '(∀ i, (x i).1 ≠ 0)' and '(∀ i, ubiquitous W (x i))'. These are trivial because:
- '(x i).1 = 1 ≠ 0'.
- 'ubiquitous W (x i)' follows from 'False'.

However, Lean will not actually ask for 'ubiquitous W (x i)' in the proof because we already derived 'False' earlier. But we must ensure 'x' is defined and uses the correct type.

Step-by-Step Abstract Plan

1. Derive 'False' from 'hW'':
   - Since 'N > 2 ^ n.1 ⩾ 2', 'N ⩾ 3'.

```
  – 'hW' 0' requires '0 < N', which is 'Nat.pos_of_ne_zero (by linarith)'.
  – But 'Function.Periodic W 0' is true, so 'hW' 0 ...' is 'False'.
2. Since we have 'False', construct any trivial 'x' (e.g., constant embedding to '⟨1, fun _ => 0⟩').
3. The conditions on 'x' are vacuously true ('False → anything').

### Lean 'have' Statements

'''lean4
theorem imosl_2011_c6 (W : ℤ → Fin 2) (n : ℕ+) (N : ℕ) (hN : 2 ^ n.1 < N)
  (hW : Function.Periodic W N) (hW' : ∀ N' < N, ¬ Function.Periodic W N') :
  ∃ (x : Fin n ↪ (Σ k, Fin k → Fin 2)), (∀ i, (x i).1 ≠ 0) ∧ (∀ i, ubiquitous W (x i)) := by
 -- Step 1: Derive False from hW'
 have h_false : False := by sorry
 -- Step 2: The conclusion follows trivially from h_false, but Lean needs an explicit x
 have h_main : ∃ (x : Fin n ↪ (Σ k, Fin k → Fin 2)), (∀ i, (x i).1 ≠ 0) ∧ (∀ i, ubiquitous W (x i)) := by sorry
 sorry
'''

### Complete Lean 4 Proof

'''lean4
theorem imosl_2011_c6 (W : ℤ → Fin 2) (n : ℕ+) (N : ℕ) (hN : 2 ^ n.1 < N)
  (hW : Function.Periodic W N) (hW' : ∀ N' < N, ¬ Function.Periodic W N') :
  ∃ (x : Fin n ↪ (Σ k, Fin k → Fin 2)), (∀ i, (x i).1 ≠ 0) ∧ (∀ i, ubiquitous W (x i)) := by
 have h_false : False := by
  have h₁ : 0 < N := by
   have h₂ : 2 ^ n.1 ≥ 2 ^ 1 := by
    apply Nat.pow_le_pow_of_le_right
    · norm_num
    · have h₃ : n.1 ≥ 1 := by
      exact Nat.one_le_of_lt n.2
     linarith
   have h₃ : N > 2 ^ n.1 := hN
   have h₄ : N > 0 := by linarith
   linarith
  have h₂ : ¬Function.Periodic W 0 := by
   apply hW' 0
   <;> simpa [h₁] using hN
  have h₃ : Function.Periodic W 0 := by
   intro z
   simp [Function.Periodic]
  exact h₂ h₃

 have h_main : ∃ (x : Fin n ↪ (Σ k, Fin k → Fin 2)), (∀ i, (x i).1 ≠ 0) ∧ (∀ i, ubiquitous W (x i)) := by
  exfalso
  exact h_false

 exact h_main
'''
```

D. Revision to MiniF2F

1. mathd_algebra_247:

```
/-- Let t = 2s − s² and s = n² − 2ⁿ + 1. What is the value of t when n = 3? Show that it is 0.-/
theorem mathd_algebra_247 (t s : ℝ) (n : ℤ) (h₀ : t = 2 * s − s ^ 2) (h₁ : s = n ^ 2 − 2 ^ n + 1)
  (n) (_ : n = 3) : t = 0 := by
 sorry
-- revise to
```

```
theorem mathd_algebra_247 (t s : ℝ) (n : ℤ) (h₀ : t = 2 * s - s ^ 2) (h₁ : s = n ^ 2 - 2 ^ n + 1)
  (_ : n = 3) : t = 0 := by
 sorry
```

2. `induction_sum_odd`:

```
/-- Show that for positive integer n, ∑_{k=0}^{n-1} (2k + 1) = n^2.-/
theorem induction_sum_odd (n : ℕ) : (∑ k in Finset.range n, 2 * k) + 1 = n ^ 2 := by
 sorry
-- revise to
theorem induction_sum_odd (n : ℕ) : (∑ k in Finset.range n, (2 * k + 1)) = n ^ 2 := by
 sorry
```

3. `induction_prod1p1onk3le3m1onn`:

```
/-- Show that for any positive integer n, we have ∏_{k=1}^{n} (1 + 1/k^3) ≤ 3 - 1/n.-/
theorem induction_prod1p1onk3le3m1onn (n : ℕ) (h₀ : 0 < n) :
  (∏ k in Finset.Icc 1 n, 1 + (1 : ℝ) / k ^ 3) ≤ (3 : ℝ) - 1 / ↑n := by
 sorry
-- revise to
theorem induction_prod1p1onk3le3m1onn (n : ℕ) (h₀ : 0 < n) :
  (∏ k in Finset.Icc 1 n, (1 + (1 : ℝ) / k ^ 3)) ≤ (3 : ℝ) - 1 / ↑n := by
 sorry
```