Prover Agent: An Agent-based Framework for Formal Mathematical Proofs

Kaito Baba [1] **Chaoran Liu** [2] **Shuhei Kurita** [3 2] **Akiyoshi Sannai** [4 5 6 2 7]

Abstract

We present Prover Agent, a novel AI agent for automated theorem proving that integrates large language models (LLMs) with a formal proof assistant, Lean. Prover Agent coordinates an informal reasoning LLM, a formal prover model, and feedback from Lean while also generating auxiliary lemmas to assist in discovering the overall proof strategy. It achieves an 86.1% success rate on the MiniF2F benchmark, establishing a new state-of-the-art among methods using small language models (SLMs) with a much lower sample budget than previous approaches. We also present case studies illustrating how these generated lemmas contribute to solving challenging problems.

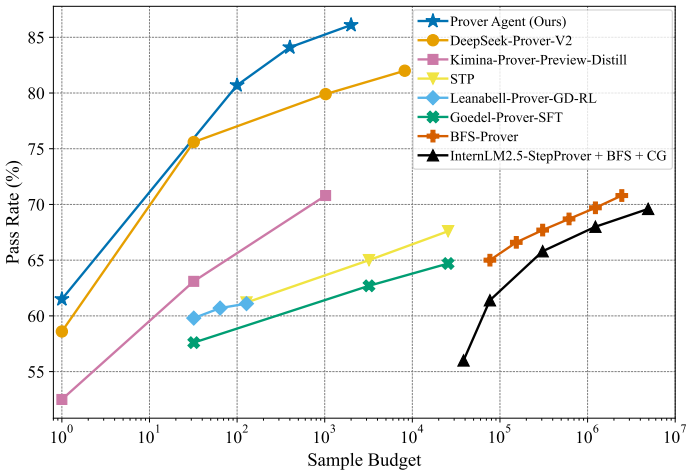

Figure 1. Comparison of theorem-proving performance on the MiniF2F benchmark (Zheng et al., 2022) among methods using SLMs. Our approach achieves a higher success rate with fewer sample budgets, establishing a new state-of-the-art at this scale.

1. Introduction

Recent advances in the reasoning capabilities of large language models (LLMs) have driven remarkable progress across many areas of artificial intelligence, including mathematical theorem proving and problem solving (OpenAI, 2024b; DeepSeek-AI, 2025; Yang et al., 2025a). However, LLMs are prone to errors and hallucinations that can undermine their reliability (Ji et al., 2023; Huang et al., 2025; Xu et al., 2025). Inference-time scaling techniques such as chain-of-thought have greatly enhanced their reasoning performance by allowing models to reflect on and correct faulty reasoning steps (OpenAI, 2024b; DeepSeek-AI, 2025; Wei et al., 2022). Nonetheless, eliminating mistakes entirely remains challenging, especially for more difficult problems (Wei et al., 2022; Zeng et al., 2025).

Formal proof assistants such as Lean (Moura & Ullrich, 2021), The Rocq Prover (previously known as Coq) (Barras et al., 1999), and Isabelle (Paulson, 1994) rigorously verify by computer that every inference step in mathematical proofs written in their respective languages is correct, based on the Curry–Howard correspondence. This helps mathematicians verify the correctness of proofs. Here, no errors, omissions of detail, implicit assumptions, or ambiguities are permitted. However, working with formal proof assistants typically requires painstaking manual effort and meticulous detail. As a result, automating mathematical theorem proving has long been a grand challenge in artificial intelligence and formal methods (Newell & Simon, 1956; Irving et al., 2016; Polu & Sutskever, 2020a; Jiang et al., 2023; Lu et al., 2023).

Consequently, formal theorem proving with LLMs has become increasingly important in recent years, leading to a growing body of research in this area (Wang et al., 2024b; Wu et al., 2024a; Xin et al., 2025b; Li et al., 2025; Xin et al., 2025a; Dong & Ma, 2025; Lin et al., 2025; Zhang et al., 2025; Wang et al., 2025; Ren et al., 2025). This not only provides a way to guarantee the correctness of mathematical reasoning by LLMs, but also marks a major breakthrough in automated theorem proving. A key point is the complementary strengths of LLMs and formal proof assistants: LLMs excel in reasoning and generation but may produce errors

[1]The University of Tokyo, Tokyo, Japan [2]Research and Development Center for Large Language Models, National Institute of Informatics, Tokyo, Japan [3]National Institute of Informatics, Tokyo, Japan [4]Kyoto University, Kyoto, Japan [5]Shiga University, Shiga, Japan [6]RIKEN Center for Advanced General Intelligence for Science Program, Kobe, Japan [7]National Institute of Science Technology Policy (NISTEP), Tokyo, Japan. Correspondence to: Kaito Baba <baba-kaito662@g.ecc.u-tokyo.ac.jp>, Akiyoshi Sannai <sannai.akiyoshi.7z@kyoto-u.ac.jp>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.

and lack guarantees of correctness, whereas formal proof assistants, such as Lean, possess perfect verification capabilities grounded in mathematical logic but are not generative.

Yet, significant hurdles remain in bridging informal reasoning and formal proving (Yang et al., 2025b). For instance, prompting o3-mini (OpenAI, 2025) to directly generate a complete Lean proof for a competition-level problem succeeds in only 6.0% of cases in a single attempt, despite its strong performance on competition-level mathematical reasoning in natural language (Yousefzadeh & Cao, 2025). Even when fine-tuned on mathematical data, trained with reinforcement learning, or allowed chain-of-thought, purely neural approaches fail to produce correct formal proofs, and their formal proving capabilities still lag far behind their informal reasoning skills in natural language (Yang et al., 2025b; OpenAI, 2025; Ren et al., 2025).

To bridge this gap between informal reasoning and formal proving, we propose a novel agent framework (**Prover Agent**) that coordinates an informal reasoning LLM, a formal prover model, and the Lean verification system. To tackle difficult problems that cannot be solved directly, the agent generates auxiliary lemmas to assist in discovering a viable proof strategy, much like how humans approach problems when the initial proof direction is unclear. On the MiniF2F benchmark (Zheng et al., 2022), it achieves an 86.1% success rate, establishing a new state-of-the-art among methods using small language models (SLMs). This performance is on par with that of an International Mathematical Olympiad (IMO) gold medalist. Notably, it uses only SLMs with much smaller sample budget than previous high-performing approaches, making it much more efficient in terms of inference-time cost.

Our contributions are summarized as follows:

- **Coordination of Informal and Formal Reasoning with Lean Feedback:** Our agent combines an informal LLM and a formal prover under Lean's verification. The LLM produces natural language reasoning and lemmas, which the prover formalizes and Lean checks. Errors detected by Lean are immediately fed back, enabling iterative refinement of constructed proofs.
- **Auxiliary Lemma Generation for Strategy Discovery:** For challenging problems that cannot be solved directly, our agent generates auxiliary lemmas, such as specific cases, intermediate facts, or hypothesis-driven conjectures, which are then formally proved. By reconsidering the overall proof in light of the verified lemmas, the system uncovers viable proof strategies even when the solution path is not apparent at first.
- **State-of-the-Art Theorem-Proving Performance:** On the challenging MiniF2F benchmark (Zheng et al., 2022), a standard benchmark for formal theorem proving that consists of 488 problems drawn from mathematics Olympiads and advanced mathematics, our agent achieves 86.1% pass rate, establishing a new state-of-the-art among methods using SLMs.
- **Efficiency in Inference-Time Cost:** The 86.1% success rate was achieved using only SLMs with a much smaller sample budget than previous high-performing approaches. This emphasizes the efficiency of our approach in terms of inference-time cost.

2. Related Work

2.1. LLMs for Formal Theorem Proving

The use of language models for guiding formal theorem provers has gained momentum recently. Early work like GPT-f (Polu & Sutskever, 2020b) applied transformers to produce proofs in formal systems such as Metamath (Megill & Wheeler, 2019) and Lean (Moura & Ullrich, 2021) by generating one proof step (tactic) at a time, guided by a goal state. Subsequent efforts in Lean, e.g. lean-gptf and PACT (Han et al., 2022), fine-tuned LLMs on large corpora of proof data, achieving moderate success in automatically discovering proofs (Polu et al., 2023a; Wu et al., 2024a). However, these models often operated in a step-by-step tactic prediction mode, requiring complex search algorithms like best-first search or Monte Carlo Tree Search (MCTS) to explore different proof paths (Lample et al., 2022; Wang et al., 2023; Wu et al., 2024a; Li et al., 2025; Xin et al., 2025b). This yielded some notable results but at high computational cost and without fully leveraging the LLM's ability for high-level planning.

Another line of work attempted whole-proof generation by having the model output an entire proof script in one go, rather than incrementally (First et al., 2023; Xin et al., 2025a; Lin et al., 2025; Zhang et al., 2025). For instance, the Goedel-Prover (Lin et al., 2025) and Leanabell-Prover (Zhang et al., 2025) approached theorem proving by training on full proof examples. These "one-shot" proof generators can sometimes solve easy problems quickly, but they struggle with longer, deeper proofs due to the LLM's difficulty in maintaining logical consistency over long outputs. Indeed, recent studies have found that as proofs grow in length and complexity, LLMs "often lose track of the crucial information" and produce incomplete or logically flawed proofs (Wei et al., 2022; Zeng et al., 2025).

Other notable work includes LeanDojo (Yang et al., 2023), which augments tactic generation with the retrieval of relevant premises (facts from the math library), and COPRA (Thakur et al., 2024), which uses GPT-4 (OpenAI, 2024a) in-context to propose proof steps in Lean. These systems improved proof success by providing better guidance or context to the proving process.

2.2. Recent Advancement in Formal Reasoning

Bridging the gap between informal (natural language) mathematical reasoning and formal proof checkers is a topic of active research. Xin et al. (2025a); Ren et al. (2025) and others explored prompting an LLM to generate informal reasoning steps (a chain-of-thought) before mapping them to formal tactics. While this showed that having intermediate reasoning can help, these approaches still relied on relatively short reasoning sequences and did not deeply integrate a feedback loop from the formal prover.

Recently, Kimina-Prover (Wang et al., 2025) has demonstrated state-of-the-art results in Lean 4 theorem proving by learning a specialized formal reasoning pattern via reinforcement learning. Trained on a large number of formal proofs and guided with human-like reasoning heuristics, it achieved an 80.7% pass rate on MiniF2F (Zheng et al., 2022) with a 72B-parameter model. Notably, their approach implicitly learns to perform longer chains of reasoning and is likely to "flatten" the search process by deciding what intermediate steps to prove. Our method can be seen as a complementary, modular approach, where instead of training one massive model end-to-end, we leverage a powerful pre-trained LLM and an existing prover and coordinate them. This design is inspired by the neuro-symbolic paradigm, treating the LLM and the prover as two specialized "agents."

More recently, DeepSeek-Prover-V2 (Ren et al., 2025) pushed this paradigm further, demonstrating new state-of-the-art results in Lean 4 theorem proving. In the DeepSeek-Prover-V2 pipeline, DeepSeek-V3 (DeepSeek-AI, 2024) is first used to generate proof sketches and define subgoals; each subgoal is then formalized into Lean 4 fragments, which are substituted into an initial proof sketch to form a complete proof. The completed proof was then employed as cold-start data for reinforcement learning. DeepSeek-Prover-V2 achieved an 88.9% pass rate on MiniF2F with a 671B-parameter model, surpassing the performance of Kimina-Prover. The subgoal decomposition approach in DeepSeek-Prover-V2 and other earlier works (Jiang et al., 2023; Wang et al., 2024a) shares certain similarities with ours, but our method adopts a more comprehensive strategy that subsumes it. In these works, the full sketch of the proof must be correctly envisioned upfront, which is often challenging. In contrast, our approach does not assume that the overall proof strategy is fully visible from the beginning. Rather than limiting decomposition to subgoals directly aligned with a pre-defined proof plan, we also consider auxiliary lemmas, such as specific cases or intermediate facts, that may aid in developing a strategy. This approach enables us to solve problems whose complete structure was not apparent at first.

3. Method

Our theorem-proving agent consists primarily of four components: a LLM specialized in informal reasoning in natural language; a prover model trained for formal proving in Lean; a formal proof assistant, Lean; and a bridging model that transforms lemmas generated in natural language into formal statements, which formalizes only their assumptions and conclusions without attempting proofs at this stage.

The overall workflow is illustrated in Figure 2 and the corresponding pseudocode is shown in Algorithm 1. Given a formal math problem, our agent begins by attempting to directly prove it, since this is often sufficient for simpler problems. For more difficult problems that cannot be solved directly, it generates auxiliary lemmas that may help uncover a viable proof strategy. These lemmas are then formalized and proved individually, and the resulting proven lemmas are used to synthesize a final proof of the original problem. Throughout this process, feedback from Lean is used to iteratively refine constructed proofs, ensuring syntactic and logical correctness at every step.

We describe each stage below, highlighting how the informal LLM, formal prover model, and Lean coordinate to construct formal proofs.

3.1. Formal Proof Construction Guided by Informal Reasoning and Iterative Feedback

We first describe the direct proving approach without decomposition. This approach is used for the initial attempt to prove the given problem directly and for proving the generated lemmas.

In order to leverage the stronger mathematical reasoning ability of the informal LLM compared to that of the formal prover model, we first generate an informal proof in natural language for the given problem or lemma using the informal LLM. This proof is not yet formalized but provides a more precise mathematical line of reasoning. Then, the formal prover model utilizes the informal proof as contextual guidance to generate a formal proof. The generated formal proof is then verified by Lean, checking that it is logically correct and adheres to the formal rules of the system. If the proof is successful, this step is complete. In the case of a given problem, the generated proof serves as the final result. In the case of a generated lemma, the proven lemma is stored, and we move on to the next lemma or the final proof assembly. If the proof fails, these steps are repeated until a successful proof is found or the maximum number of attempts N_{init} is reached. This process helps establish a better initial outline for the subsequent iterative refinement process.

If the proof still fails, the agent enters an iterative refinement phase. The proof with the minimal number of Lean verification errors among the prior attempts is selected as

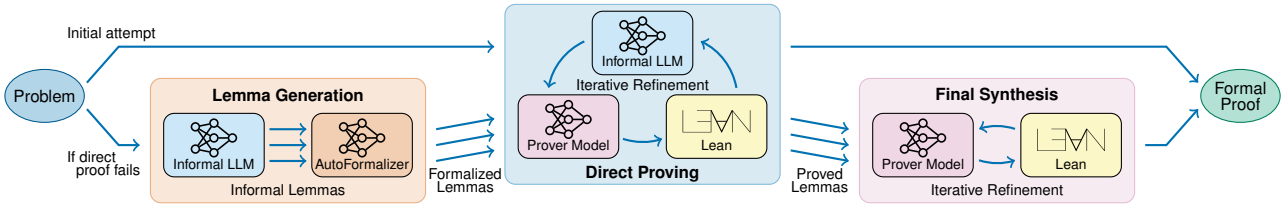

Figure 2. Overall workflow of our agent. Our agent coordinates informal reasoning, formal proving, and verification by Lean. It first attempts direct proving; if unsuccessful, it generates auxiliary lemmas to support the discovery of a viable proof strategy. These lemmas are then formally proved. Finally, using only the lemmas that were successfully proved, the agent reconsiders the overall proof and synthesizes the final proof.

the initial draft. This proof is then iteratively refined based on the feedback provided by Lean, which includes error messages or information about the failure, such as "goal unsolved" or "tactic failed" at a certain step. Specifically, the previous proof attempt, along with the locations of Lean errors and the corresponding error messages, is provided to the prover model, and it rethinks and generates a corrected version of the proof. This process is repeated until the proof is successfully verified by Lean or the maximum number of attempts N_{refine} is reached. If the proof is still unsuccessful, the system either proceeds to the decomposition step or, in the case of a lemma, considers it unprovable and moves on to the next lemma.

This iterative refinement process leverages Lean's ability to verify the correctness of each proof step. Unlike pure LLM reasoning, which may generate flawed reasoning steps without detection, any mistake is immediately identified and corrected accordingly. This addresses a key limitation of inference-time scaling with chain-of-thought, where the model's limited self-correction ability becomes a bottleneck, preventing efficient solutions to difficult problems (Zeng et al., 2025; Song et al., 2025; Stechly et al., 2025).

It is accessible if a generated lemma cannot be proven. This mirrors how human mathematicians often approach problems: when the overall strategy is unclear at the beginning, they may explore several directions, some of which turn out to be unproductive and are eventually discarded in favor of more promising ones. Alternatively, to handle cases where the lemma is still too challenging to prove, the system may recursively generate the lemma into smaller, more manageable sub-lemmas up to a specified depth limit D. Also, if no generated lemma can be proven, new lemmas are generated and the process restarts, as long as the number of attempted lemmas does not exceed the predefined limit L.

3.2. Lemma Generation via Informal Reasoning

When the direct proving approach fails to solve the problem, the agent generates several auxiliary lemmas. These are not limited to subgoals that can be directly inserted into a formal proof; they may also include specific cases or potentially useful intermediate facts that can be derived from the assumptions, which may help in developing a proof strategy. This represents a key difference from prior work, which typically relies on inserting subgoals into a predefined proof sketch (Jiang et al., 2023; Wang et al., 2024a; Ren et al., 2025). In such approaches, it is necessary to come up with the correct overall proof strategy beforehand, which is often a challenging task. In contrast, our approach does not assume that the proof strategy is visible from the outset. Instead, the agent first proposes a variety of lemmas, including those that may help clarify the structure of the problem, such as specific cases or derivable facts. Only the lemmas that are successfully verified by Lean are retained and reconsidered when revisiting the overall proof. This allows the system to gradually form a proof strategy, even in cases where the full structure was not initially apparent.

For example, when trying to prove that $n^2 + an$ is even for a natural number n and an odd number a, it may be helpful to first consider specific cases such as $a = 1$ or $a = 3$, i.e., $n^2 + n$ or $n^2 + 3n$. These specific cases can help reveal patterns and guide the overall proof strategy for $n^2 + an$, even though expressions like $n^2 + n$ or $n^2 + 3n$ may not explicitly appear as steps within the final proof.

This approach mirrors how human mathematicians typically work. When the overall strategy is not clear at the beginning, they often explore specific cases or consider what can be derived from the assumptions. Through such trial and error, they gradually discover the overall proof strategy.

The generation of lemmas is first carried out in natural language, i.e., the system initially produces each lemma in natural language. This is because the informal reasoning LLM has stronger mathematical reasoning capabilities. Once these lemmas have been generated in natural language, they are then converted into formal form using an LLM specialized in autoformalization, such as Kimina-Autoformalizer-7B (Wang et al., 2025). Note that at this stage, only the assumptions and conclusions of the lemmas are formalized; their proofs are not yet constructed, and a `sorry` placeholder is inserted instead. Here, too, Lean is used to verify the syntactic correctness of the formalized statements, and they are regenerated iteratively until they become syntactically valid. These formally stated lemmas are then

Algorithm 1 The overall architecture of our lemma-based theorem-proving agent coordinating informal reasoning, formal reasoning, and Lean.

Input: Problem T with hyperparameters N_{init} (max initial proof attempts) and N_{refine} (max refinement attempts)
Output: Formal proof of T or `failure`

function MAIN(T): Overall proof process for problem T
  $P_{\text{direct}} \leftarrow$PROVE($T$): Attempt to prove theorem T directly
  if P_{direct} succeeds **then**
    return P_{direct}
  end if
  // Generate lemmas
  Informal LLM generates lemmas $L_1, L_2, \ldots, L_n$ in natural language
  for each lemma L_i **do**
    AutoFormalizer converts L_i into Lean statement F_i
    Lean checks F_i. If failing, regenerate F_i until syntactically correct
  end for
  // Prove each lemma
  for each lemma F_i **do**
    $P_i \leftarrow$PROVE(F_i): Attempt to prove lemma F_i
  end for
  // Collect proven lemmas
  $\mathcal{P}_{\text{proven}} \leftarrow \{P_i \mid P_i \text{ is succeeded}\}$
  // Synthesize final proof using proven lemmas
  for $k = 1$ to N_{init} **do**
    $P_{\text{final}} \leftarrow$ Prover synthesizes proof of T using $\mathcal{P}_{\text{proven}}$
    Lean checks P_{final}
    if the check succeeds **then**
      return P_{final}
    end if
  end for
  // Iterative refinement of final proof
  $P_{\text{best}} \leftarrow$ Best previous proof attempt with the fewest Lean errors
  return ITERATIVEREFINE(P_{best})
end function

function PROVE(S): Attempt to generate an informal proof of S
  // Initial proof attempt
  for $k = 1$ to N_{init} **do**
    Informal LLM generates informal proof P_{inf} of S
    Prover attempts to formalize P_{inf} into P_{form}
    Lean checks P_{form}
    if the check succeeds **then**
      return P_{form}
    end if
  end for
  // Iterative refinement
  $P_{\text{best}} \leftarrow$ Best previous proof attempt with the fewest Lean errors
  return ITERATIVEREFINE(P_{best})
end function

function ITERATIVEREFINE(P): Refine proof P based on Lean feedback
  for $k = 1$ to N_{refine} **do**
    Prover generates revised proof P' based on Lean feedback
    Lean checks P'
    if the check succeeds **then**
      return P'
    else
      $P \leftarrow P'$ // Update best proof
    end if
  end for
  return `failure` // No proof found after max attempts
end function

proved using the proof construction process described in Section 3.1, where the proof process is guided by informal reasoning along with feedback from Lean.

3.3. Final Proof Synthesis Guided by Verified Lemmas and Iterative Feedback

After attempting to prove each of these lemmas individually, the agent reconsiders the overall proof using only lemmas that have been formally verified, enabling it to solve problems whose complete structure was not apparent at first. Specifically, only the lemmas that have been successfully verified are included as context, and the prover is invoked again to generate a proof of the original problem.

As in Section 3.1, Lean feedback is utilized in this stage as well. The prover first attempts to construct a complete proof while referring to the verified lemmas, repeating this process for a fixed number of iterations N_{init}. If a valid proof is still not found after these attempts, the system enters the iterative refinement phase. Among the previously generated proofs, the one with the fewest Lean errors is selected as the initial outline. The prover model is then provided with the failed proof, the locations of the errors, and the corresponding error messages, and it generates a revised version accordingly. This process is repeated until a valid proof is found or the maximum number of attempts N_{refine} is reached.

4. Experiments

We perform experiments to evaluate the effectiveness of our approach for formal theorem proving. We compare our approach against various existing methods, including those representing the current state-of-the-art in formal theorem proving. We also conduct case studies to illustrate how auxiliary lemmas help uncover proof strategies for challenging problems based on these experiments.

4.1. Experimental Setup

Benchmarking Dataset. We evaluate our approach on the MiniF2F benchmark (Zheng et al., 2022), a standard dataset for evaluating formal theorem-proving systems. The detailed description of the dataset and the way we use it are provided in Appendix A.1.

Used Models. For the informal reasoning LLM, we use `DeepSeek-R1-0528-Qwen3-8B` (DeepSeek-AI, 2025). For the formal prover model, we use `DeepSeek-Prover-V2-7B` (Ren et al., 2025), a state-

of-the-art model for formal proving in Lean at the 7B scale. For the autoformalization step, we use `Kimina-Autoformalizer-7B` (Wang et al., 2025), a 7B model trained to convert natural language statements into Lean 4 code with `sorry` placeholders for proofs. See Appendix A.2 for more details.

Implementation Details. All models are invoked via vLLM (Kwon et al., 2023). Further implementation details are provided in Appendix A.3.

Sample Budget. We set the maximum number of initial proof attempts N_{init} to 100, meaning the prover model is allowed to generate up to 100 different proof attempts for each lemma or theorem. We set the maximum number of refinement attempts N_{refine} to 300, meaning the prover model is allowed to refine the proof up to 300 times. Note that these numbers are much lower than those in prior work. Even when summing over all stages in our pipeline, the total number of attempts in our settings remains far below the 8,292 attempts required to reach high success rates in prior work (Wang et al., 2025; Ren et al., 2025). Detailed discussion of the sample budget is provided in Section 5.2. We set the maximum lemma generation depth D to 1, meaning no further lemmas are generated for the lemmas.

Baseline Methods. We compare our approach against several baseline methods. See Appendix A.4 for more details. The baseline results are sourced from the original papers.

5. Results and Discussion

The results are shown in Table 1 and Figure 1. Our lemma-based agent achieves an 86.1% success rate (210 out of 244 problems solved), establishing a new state-of-the-art among methods using small language models (SLMs).

5.1. Comparison with the Previous State-of-the-Art

Compared to the baseline method, our approach, which achieves an 86.1% success rate, outperforms DeepSeek-Prover-V2-7B (Ren et al., 2025), which was the previous state-of-the-art among small models with an 82.0% success rate, demonstrating its effectiveness.

5.2. High Success Rate under Low Sample Budget

Since we set $N_{\text{init}} = 100$ and $N_{\text{refine}} = 300$, the sample budgets for direct proving without and with iterative refinement are 100 and 400, respectively. The overall proving pipeline consists of three stages: direct proving, proving for each generated lemma, and the final proof synthesis based on the proven lemmas. Since we set the maximum number of lemmas to be generated to 3, the total sample budget is $(1 + 3 + 1) \times 400 = 2000$ per problem.

This sample budget is much lower than the 8192 samples required by DeepSeek-Prover-V2-7B (Ren et al., 2025) and many other previous methods to achieve a high pass rate, showing the efficiency of our approach.

5.3. Effectiveness of Informal, Formal, and Lean Coordination

With a sample budget of just 1, our agent achieves a 61.5% success rate, surpassing DeepSeek-Prover-V2-7B's 58.6%. When the sample budget is increased to 100, the success rate rises to 80.7%, and with iterative refinement (sample budget 400), it reaches 83.1%. Remarkably, both results surpass DeepSeek-Prover-V2-7B's 79.9% success rate achieved with a much larger sample budget of 1,024. This demonstrates the effectiveness of our agent's coordination between informal reasoning in natural language and formal feedback from Lean in constructing formal proofs.

Even without iterative refinement, coordinating with informal reasoning alone improves performance under a much smaller sample budget. This suggests that the coordination effectively leverages the stronger mathematical reasoning ability of the informal LLM compared to formal provers, bridging the gap between informal and formal reasoning.

The interplay between the LLM and Lean is also important. While the LLM on its own would make many mistakes, as evidenced by the low direct success rate, the immediate feedback from lean turns these errors into opportunities for correction. Our refinement loop can be seen as a form of self-correction through in-context learning, akin to how humans improve their understanding based on feedback. This provides an efficient approach to a key limitation of inference-time scaling with chain-of-thought, where simply increasing the number of reasoning steps does not guarantee better results due to the model's limited ability of self-correction (Zeng et al., 2025; Song et al., 2025; Stechly et al., 2025).

5.4. Case Study: Success with Lemma-Guided Proofs

Here, we present a case study where incorporating lemmas enabled the agent to synthesize proof successfully. We analyze in detail the reasoning process for the problem `induction_nfactltnexpnm1ngt3`, a case where the direct proof attempt failed but the use of auxiliary lemmas led to a successful proof. This problem asks for a formal proof that, for all natural numbers $n > 3$, the inequality $n! < n^{n-1}$ always holds. The outputs for this problem, such as the generated lemmas, final formal proof, and the associated reasoning process, are provided in Appendix B.

In this case, the agent generated the following three lemmas: The first states that $3! < 3^{3-1}$; the second states that for any natural number $n \geq 2$, $n^{n-1} < (n+1)^{n-1}$; and the third states that for any natural number $n \geq 3$, $n! < (n+1)^{n-1}$. The first is a specific case of the original problem with $n = 3$, while the second may provide a helpful hint toward

Table 1. Comparison of formal theorem-proving performance on miniF2F-test. The results are reported as the percentage of theorems proved correctly. The model size is given in billions of parameters. Sample budget refers to the total number of proof attempts allowed for a system to solve a given problem. For our agent, it includes all proof attempts across the full pipeline, including initial direct proving, iterative refinement, lemma proving, and final proof synthesis. The best results within each model scale are highlighted in **bold**.

Prover System	Method	Model Size	Sample Budget	miniF2F-test
Large Language Models				
Kimina-Prover-Preview (Wang et al., 2025)	Whole-proof	72B	1	52.9%
			32	68.9%
			1024	77.9%
			8192	80.7%
DeepSeek-Prover-V2 (non-CoT) (Ren et al., 2025)	Whole-proof	671B	1	59.5%
			32	73.8%
			1024	76.7%
			8192	78.3%
DeepSeek-Prover-V2 (CoT) (Ren et al., 2025)	Whole-proof	671B	1	61.9%
			32	82.4%
			1024	86.6%
			8192	**88.9%**
Small Language Models				
DeepSeek-Prover-V1.5-RL + RMaxTS (Xin et al., 2025a)	Tree search	7B	$32 \times 16 \times 400$	63.5%
InternLM2.5-StepProver + BFS + CG (Wu et al., 2024a)	Tree search	7B	$256 \times 32 \times 600$	65.9%
HunyuanProver v16 + BFS + DC (Li et al., 2025)	Tree search	7B	$600 \times 8 \times 400$	68.4%
BFS-Prover (Xin et al., 2025b)	Tree search	7B	$2048 \times 2 \times 600$	70.8%
Leanabell-Prover-GD-RL (Zhang et al., 2025)	Whole-proof	7B	128	61.1%
Goedel-Prover-SFT (Lin et al., 2025)	Whole-proof	7B	25600	64.7%
STP (Dong & Ma, 2025)	Whole-proof	7B	25600	67.6%
Kimina-Prover-Preview-Distill (Wang et al., 2025)	Whole-proof	7B	1	52.5%
			32	63.1%
			1024	70.8%
DeepSeek-Prover-V2 (non-CoT) (Ren et al., 2025)	Whole-proof	7B	1	55.5%
			32	68.0%
			1024	73.2%
			8192	75.0%
DeepSeek-Prover-V2 (CoT) (Ren et al., 2025)	Whole-proof	7B	1	58.6%
			32	75.6%
			1024	79.9%
			8192	82.0%
Prover Agent (Ours) (Direct proving w/o iterative refinement)	Agent	8B	1	61.5%
(Direct proving w/o iterative refinement)			100	80.7%
(Direct proving w/ iterative refinement)			400	84.0%
(Final Proof Synthesis w/ Lemma)			2000	**86.1%**

solving the original problem. Both were easily proven in a single direct proof attempt. The third lemma generated in this case asserts that for any natural number $n \geq 3$, $n! < (n+1)^{n-1}$. This lemma closely resembles the original problem, as it is a slightly weaker version of its conclusion. Due to its similarity and retained difficulty, the agent failed to construct a direct proof for it.

By examining the final successful reasoning trace in Appendix B.3, we see that the specific case for $n = 3$, considered as the first lemma, appears explicitly on line 7. The reasoning also checks the cases for $n = 4$ and $n = 5$, following a similar pattern. Furthermore, as stated on line 13, the use of mathematical induction is clearly identified as the intended proof strategy. Then, the reasoning trace from line 14 to line 80 further elaborates the proof process within the framework of mathematical induction. Furthermore, in the final proof, the proof technique used in Lemma 2 is explicitly applied at lines 195–196.

Next, as a comparison, we analyze the reasoning process from the initial direct proving attempt without using any

Table 2. Comparison of formal theorem-proving performance by problem category on MiniF2F-test. The results are reported as the percentage of theorems proved correctly. The model size is given in billions of parameters. Sample budget refers to the total number of proof attempts allowed for a system to solve a given problem. The best results in each category are highlighted in **bold**.

		Model Size	Sample Budget	Olympiad				MATH			Custom			
				IMO	AIME	AMC	Sum	Algebra	Number Theory	Sum	Algebra	Number Theory	Induction	Sum
Number of Problems				20	15	45	80	70	60	130	18	8	8	34
Prover Agent (Ours)	(Direct proving w/o iterative refinement)	8B	1	40.0	53.3	62.2	55.0	71.4	60.0	66.2	55.6	75.0	50.0	58.8
	(Direct proving w/o iterative refinement)		100	70.0	80.0	82.2	78.8	82.9	88.3	85.4	66.7	75.0	62.5	67.6
	(Direct proving w/ iterative refinement)		400	80.0	80.0	88.9	85.0	84.3	91.7	87.7	66.7	75.0	62.5	67.6
	(Final Proof Synthesis w/ Lemma)		2000	**80.0**	80.0	**91.1**	**86.3**	85.7	91.7	88.5	72.2	87.5	75.0	76.5
DeepSeek-Prover-V2 (Ren et al., 2025)		671B	8192	50.0	**93.3**	77.8	73.8	**100.0**	**96.7**	**98.5**	**83.3**	**87.5**	**100.0**	**88.2**

lemmas, as shown in Appendix B.4. Here, we present the reasoning trace that resulted in the fewest Lean errors among all initial direct attempts. Compared to the successful case with lemmas, we see that the proof strategy is much less clear in this direct attempt. In the "Key Observations" section (lines 6 to 14), there is no indication of using mathematical induction, unlike in the lemma-assisted case. Although the system explores several ideas from lines 15 to 63, the reasoning appears less focused and more exploratory, lacking a concrete plan. As a result, while it eventually leans toward using induction, the lack of a clear and structured approach prevents it from working out the necessary details, ultimately leading to failure in the formal proof, which tolerates no ambiguity.

This detailed case study highlights the effectiveness of our lemma-generation approach in uncovering viable proof strategies. This marks a significant advance over prior methods that decompose problems into subgoals, which often assume the overall proof strategy is known in advance. Identifying an initial proof strategy is often a challenging part of solving difficult problems. Indeed, Ren et al. (2025) employs a decomposition-based approach but relies on the much larger and stronger DeepSeek-V3 (DeepSeek-AI, 2024) to formulate the initial proof sketch. In contrast, our agent follows a reasoning process similar to that of human mathematicians when the proof strategy is not apparent at first glance, exploring specific cases or hypothesizing intermediate steps to discover a promising direction and ultimately uncover the overall proof strategy.

5.5. Performance on Olympiad-Level Problems

Table 2 shows the results for each category on the MiniF2F-test dataset. These results demonstrate that our approach performs particularly well on Olympiad-level problems, even surpassing DeepSeek-Prover-V2 (Ren et al., 2025), which uses a significantly larger 671B model and a much higher sample budget of 8192.

Given that our direct proving method without iterative refinement and with a sample budget of only 100 already surpasses DeepSeek-Prover-V2, this suggests that coordination with natural language-based informal reasoning may be the key. Olympiad-level problems require a high degree of mathematical reasoning, and the strong reasoning abilities of the informal LLM likely played a crucial role in solving them effectively. On the other hand, our agent does not outperform DeepSeek-Prover-V2 in the MATH and Custom categories. The consistent gap in these categories suggests that model size and sample budget may play a more significant role here. Since DeepSeek-Prover-V2 also possesses a certain level of mathematical reasoning ability, it can handle these relatively mathematically easier problems on its own.

5.6. Modular and Scalable Design

Unlike monolithic approaches that rely on training a single large model end-to-end, our method takes an orthogonal approach by combining an existing LLM and formal prover, without any joint training. This modular design offers a key practical advantage: as LLMs improve, our system can immediately benefit by simply replacing components, making it easy to scale with future advancements. It separates the problem into two more manageable parts by handling informal reasoning and formal proving independently, rather than relying on a single monolithic model.

5.7. Broader Applicability and Future Potential

Nothing in our pipeline is specific to mathematics competition problems. The same approach could be applied to formal proofs in other domains, such as learning theory or physics, as long as the LLM has relevant knowledge or is provided with an appropriate knowledge base. This offers the potential for AI-driven construction of mathematical theories without hallucinations or logical errors.

6. Conclusion

We introduced Prover Agent, a modular framework that coordinates an informal reasoning LLM, a formal prover model, and Lean verification. By generating auxiliary lemmas and leveraging feedback-driven refinement, our method achieved state-of-the-art performance among SLMs on the MiniF2F benchmark.

Acknowledgements

This work was supported by the "R&D Hub Aimed at Ensuring Transparency and Reliability of Generative AI Models" project of the Ministry of Education, Culture, Sports, Science and Technology. We would also like to thank the Automated Research Project (Autores) for providing access to their API during the initial stages of this research. Additionally, part of this work was supported by Advanced General Intelligence for Science Program (AGIS), the RIKEN TRIP initiative.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

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

DeepSeek-AI. DeepSeek-V3 technical report. 2024.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

First, E., Rabe, M. N., Ringer, T., and Brun, Y. Baldur: Whole-proof generation and repair with large language models. ESEC/FSE 2023, pp. 1229–1241. Association for Computing Machinery, 2023. ISBN 9798400703270. doi: 10.1145/3611643.3616243.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

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

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2), 2025. ISSN 1046-8188. doi: 10.1145/3703155.

Irving, G., Szegedy, C., Alemi, A. A., Een, N., Chollet, F., and Urban, J. DeepMath - deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), 2023. ISSN 0360-0300. doi: 10.1145/3571730.

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

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. ISBN 9798400702297. doi: 10.1145/3600006.3613165.

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X. Hypertree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, volume 35, pp. 26337–26349. Curran Associates, Inc., 2022.

Li, Y., Du, D., Song, L., Li, C., Wang, W., Yang, T., and Mi, H. HunyuanProver: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2025.

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

Lu, P., Qiu, L., Yu, W., Welleck, S., and Chang, K.-W. A survey of deep learning for mathematical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14605–14631. Association for Computational Linguistics, July 2023. doi: 10.18653/v1/2023.acl-long.817.

Megill, N. D. and Wheeler, D. A. *Metamath: A Computer Language for Pure Mathematics*, 2019.

URL http://us.metamath.org/downloads/metamath.pdf.

Moura, L. d. and Ullrich, S. 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*, LNCS 12699, pp. 625–635. Springer-Verlag, 2021. doi: 10.1007/978-3-030-79876-5_37.

Newell, A. and Simon, H. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956. doi: 10.1109/TIT.1956.1056797.

OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2024a.

OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024b.

OpenAI. OpenAI o3-mini, January 2025. URL https://openai.com/index/openai-o3-mini/.

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

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

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

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning, 2023a.

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations*, 2023b.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. 2025.

Song, Y., Zhang, H., Eisenach, C., Kakade, S. M., Foster, D., and Ghai, U. Mind the gap: Examining the self-improvement capabilities of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Stechly, K., Valmeekam, K., and Kambhampati, S. On the self-verification limitations of large language models on reasoning and planning tasks. In *The Thirteenth International Conference on Learning Representations*, 2025.

Thakur, A., Tsoukalas, G., Wen, Y., Xin, J., and Chaudhuri, S. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024.

Wang, H., Yuan, Y., Liu, Z., Shen, J., Yin, Y., Xiong, J., Xie, E., Shi, H., Li, Y., Li, L., Yin, J., Li, Z., and Liang, X. DT-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12632–12646. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.706.

Wang, H., Xin, H., Liu, Z., Li, W., Huang, Y., Lu, J., Yang, Z., Tang, J., Yin, J., Li, Z., and Liang, X. Proving theorems recursively. In *Advances in Neural Information Processing Systems*, volume 37, pp. 86720–86748, 2024a.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., Lu, J., de Saxcé, H., Bailey, B., Song, C., Xiao, C., Zhang, D., Zhang, E., Pu, F., Zhu, H., Liu, J., Bayer, J., Michel, J., Yu, L., Dreyfus-Schmidt, L., Tunstall, L., Pagani, L., Machado, M., Bourigault, P., Wang, R., Polu, S., Barroyer, T., Li, W.-D., Niu, Y., Fleureau, Y., Hu, Y., Yu, Z., Wang, Z., Yang, Z., Liu, Z., and Li, J. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025. doi: 10.48550/arXiv.2504.11354.

Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. TheoremLlama: Transforming general-purpose LLMs into lean4 experts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11953–11974. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.emnlp-main.667.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, 2020.

Wu, Z., Huang, S., Zhou, Z., Ying, H., Wang, J., Lin, D., and Chen, K. InternLM2.5-StepProver: Advancing automated theorem proving via expert iteration on large-scale lean problems. 2024a.

Wu, Z., Wang, J., Lin, D., and Chen, K. LEAN-GitHub: Compiling github lean repositories for a versatile lean prover. *arXiv preprint arXiv:2407.17227*, 2024b.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2025a.

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

Xu, Z., Jain, S., and Kankanhalli, M. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*, 2025.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report. 2025a.

Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612, 2023.

Yang, K., Poesia, G., He, J., Li, W., Lauter, K. E., Chaudhuri, S., and Song, D. Position: Formal mathematical reasoning—a new frontier in AI. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025b.

Yousefzadeh, R. and Cao, X. A lean dataset for international math olympiad: Small steps towards writing math proofs for hard problems. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Zeng, Z., Cheng, Q., Yin, Z., Zhou, Y., and Qiu, X. Revisiting the test-time scaling of o1-like models: Do they truly possess test-time scaling capabilities? 2025.

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

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

A. Detailed Experimental Setup

A.1. Benchmark Datasets: MiniF2F (Zheng et al., 2022)

MiniF2F (Zheng et al., 2022) consists of 488 mathematical problems formalized in Lean. These problems originate from sources such as AIME (American Invitational Mathematics Examination), AMC (American Mathematics Competitions), and IMO (International Math Olympiad) competitions, along with selected problems from the MATH dataset (Hendrycks et al., 2021), covering topics such as algebra, number theory, geometry, and analysis. Each problem is given as a Lean theorem statement. The benchmark is split into 244 validation and 244 test problems. We use the validation set during development (e.g., for tuning prompt formats) and report the final results on the test set. We use the revised version of miniF2F released by Wang et al. (2025); Ren et al. (2025).

Also, we observed that for problem names like `algebra_2varlineareq_fp3zeq11_3tfm1m5zeqn68_feqn10_zeq7`, the LLM often struggled to reliably reproduce the latter part of the name due to its unintelligible character sequence. Therefore, we modified such problem names by removing the less interpretable suffixes and replacing them with simpler, more memorable labels such as `algebra` for our experiments.

A.2. Used Models

For the informal reasoning LLM, we use `DeepSeek-R1-0528-Qwen3-8B`[1] (DeepSeek-AI, 2025), a model obtained by distilling the chain-of-thought outputs of DeepSeek-R1-0528 (DeepSeek-AI, 2025) into the Qwen3 8B Base (Yang et al., 2025a). This model surpasses Qwen3 8B Base on the AIME benchmark for natural language reasoning and achieves state-of-the-art performance at the 8B scale. For the formal prover model, we use `DeepSeek-Prover-V2-7B`[2] (Ren et al., 2025), a state-of-the-art model for Lean 4 formal proving at the 7B scale, obtained by distilling from DeepSeek-Prover-V2-671B (Ren et al., 2025). For the autoformalization step, we use `Kimina-Autoformalizer-7B`[3] (Wang et al., 2025), a 7B model trained to convert natural language statements into Lean 4 code with `sorry` placeholders for proofs. All of them are publicly available on Hugging Face (Wolf et al., 2020).

A.3. Implementation Details

The informal LLM, formal prover model, and autoformalizer are all invoked via vLLM (Kwon et al., 2023), a high-performance inference engine for large language models. We set `max_num_batched_tokens` and `max_model_len` parameters to 16384 to accommodate the long context lengths required for theorem proving, while keeping all other settings at their vLLM defaults. The models are run on NVIDIA A100 GPUs with 40GB of memory. We use Lean version 4.9.0 (Moura & Ullrich, 2021) throughout all experiments, following the same setup in Xin et al. (2025a); Ren et al. (2025).

A.4. Baseline Methods

We compare our approach against several baseline methods, categorized into two main classes: tree search methods and whole-proof generation methods. Tree search methods construct proofs incrementally by predicting individual tactics step by step, often guided by search algorithms such as best-first search or Monte Carlo Tree Search (MCTS). In contrast, whole-proof generation methods attempt to generate an entire proof script in a single forward pass, relying on the model's ability to plan the proof holistically.

The overview of the baseline methods used in our experiments is as follows:

Tree Search Method:

- **DeepSeek-Prover-V1.5-RL + RMaxTS** (Xin et al., 2025a) uses DeepSeek-Prover-V1.5-RL (Xin et al., 2025a), a 7B model trained with reinforcement learning, combined with RMaxTS (Xin et al., 2025a), a variant of MCTS that uses intrinsic rewards to explore diverse proof paths.
- **InternLM2.5-StepProver + BFS + CG** (Wu et al., 2024a) uses InternLM2.5-StepProver (Wu et al., 2024a), a 7B model trained via expert iteration (Anthony et al., 2017; Polu et al., 2023b) starting with InternLM2-StepProver (Wu

[1] https://huggingface.co/deepseek-ai/DeepSeek-R1-0528-Qwen3-8B
[2] https://huggingface.co/deepseek-ai/DeepSeek-Prover-V2-7B
[3] https://huggingface.co/AI-MO/Kimina-Autoformalizer-7B

et al., 2024b), combined with a best-first search (BFS) strategy and a critic-guided (CG) sampling technique to explore longer proofs effectively.

- **HunyuanProver v1.6 + BFS + DC** (Li et al., 2025) uses HunyuanProver, a 7B model fine-tuned via a scalable data synthesis pipeline, in conjunction with best-first search guided by the distance critic (DC) to efficiently navigate complex Lean 4 proof search spaces.

- **BFS-Prover** (Xin et al., 2025b) proposes a scalable best-first tree search framework for Lean 4 that incorporates three key innovations: strategic data filtering during expert iterations, direct preference optimization (DPO) (Rafailov et al., 2023) on state-tactic pairs using Lean compiler feedback, and length normalization to encourage exploration of deeper proof paths. BFS-Prover is a fine-tuned version of Qwen2.5-Math-7B model (Yang et al., 2024).

Whole-Proof Generation Methods:

- **Leanabell-Prover-GD-RL** (Zhang et al., 2025) is a 7B model post-trained through continual training on statement-proof pairs and reinforcement learning using Lean 4 outcome rewards. This model is a fine-tuned version of Goedel-Prover-SFT (Lin et al., 2025).

- **Goedel-Prover-SFT** (Lin et al., 2025) is a 7B-parameter model obtained by supervised fine-tuning on DeepSeek-Prover-V1.5-Base (Xin et al., 2025a) with expert-iteration.

- **STP: Self-Play Theorem Prover** (Dong & Ma, 2025) employs a self-play framework that simultaneously takes on two roles, conjecturer and prover. The conjecturer is iteratively trained on statements that are barely provable by the current prover, incentivizing it to generate increasingly challenging conjectures. The prover uses standard expert iteration to verify and prove the generated conjectures. This model is a fine-tuned version of DeepSeek-Prover-V1.5-SFT (Xin et al., 2025a), which is a 7B-parameter model.

- **Kimina-Prover-Preview** (Wang et al., 2025) is a 72B-parameter reasoning model that learns specialized formal reasoning patterns via reinforcement learning. It is pretrained on a large corpus of formal proofs and fine-tuned with a binary correctness reward and consistency penalty. They also provide **Kimina-Prover-Preview-Distill-7B**, a distilled version from the 72B model.

- **DeepSeek-Prover-V2** (Ren et al., 2025) uses DeepSeek-V3 to decompose each theorem into subgoals and then employs the proofs of those subgoals as cold-start data for reinforcement learning using binary correctness rewards and a consistency penalty to ensure that every subgoal appears in the final proof. It is implemented as a 671B-parameter model, and a distilled 7B-parameter variant is also provided.

B. Examples of Successful Cases Enabled by Lemmas

Here, we present examples where the use of auxiliary lemmas enabled successful proof construction. For detailed case studies based on these examples, see Section 5.4.

All Lean code was executed with the following header, following Xin et al. (2025a); Ren et al. (2025), which is omitted in the examples for brevity:

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat
```

Note that the following Lean code works correctly with Lean version 4.9.0, the environment used in our experiments, which strictly follows the same setup in Xin et al. (2025a); Ren et al. (2025). However, `Nat.pow_lt_pow_of_lt_left` has been deprecated in newer versions of Lean, so it must be replaced with `Nat.pow_lt_pow_left` to work with the latest environment.

B.1. The Problem

The following problem asks for a formal proof that, for all natural numbers n greater than 3, the inequality $n! < n^{n-1}$ holds:

```
theorem induction_nfactltnexpnm1ngt3
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by sorry
```

B.2. The Generated Lemmas

The generated lemmas are shown below, each presented with both its original natural language form and the corresponding Lean formalization. The first and second lemmas were successfully proved and are presented with their proofs. The third lemma could not be proved and is therefore presented with a `sorry` placeholder.

```
### Lemma 1: base_case_3
**Assumptions**:
None

**Conclusion**:
3! < 3^(3-1)
```

```
theorem base_case_3 : (3 : ℕ)! < 3^(3 - 1) := by
  have h_main : (3 : ℕ)! < 3^(3 - 1) := by
    -- Calculate the factorial and the power step-by-step
    norm_num [Nat.factorial, Nat.pow_succ, Nat.mul_assoc]
    -- Use 'decide' to confirm the inequality
    <;> decide
  exact h_main
```

```
### Lemma 2: exponent_inequality
**Assumptions**:
n is a natural number and n ≥ 2

**Conclusion**:
n^(n-1) < (n+1)^(n-1)
```

```
theorem exponent_inequality {n : ℕ} (hn : 2 ≤ n) :
    n^(n - 1) < (n + 1)^(n - 1) := by
  have h_main : n^(n - 1) < (n + 1)^(n - 1) := by
    have h₀ : n - 1 + 1 = n := by
      omega
    have h₁ : n ≥ 2 := by omega
    have h₂ : n - 1 ≥ 1 := by omega
    have h₃ : n < n + 1 := by omega
    -- Use the fact that if a < b and n > 0, then a^n < b^n
    exact calc
      n^(n - 1) < (n + 1)^(n - 1) := by
        -- Apply the lemma that if a < b and n > 0, then a^n < b^n
        exact Nat.pow_lt_pow_of_lt_left h₃ (by omega)
      _ = (n + 1)^(n - 1) := by rfl
  exact h_main
```

```
### Lemma 3: factorial_less_than_n_next_power
**Assumptions**:
n is a natural number and n ≥ 3

**Conclusion**:
n! < (n+1)^(n-1)
```

```
theorem factorial_lt_n_next_power {n : ℕ} (hn : 3 ≤ n) :
   n ! < (n + 1) ^ (n - 1) := by sorry
```

B.3. The Final Proof and Its Reasoning Trace

We present below the final Lean proof, successfully constructed by taking the previously verified lemmas into account. Alongside the code, we also include the corresponding reasoning trace generated by the prover model that guided the overall proof construction.

```
### Detailed Proof and Analysis

We are tasked with proving that for all natural numbers `n ≥ 3`, the factorial of `n`, i.e., `n!`, is less than `n^(n - 1)`.

#### Key Observations:
1. **Base Cases**:
   - For `n = 3`: `3! = 6` and `3^(3 - 1) = 3^2 = 9`. So, `6 < 9` holds.
   - For `n = 4`: `4! = 24` and `4^(4 - 1) = 4^3 = 64`. So, `24 < 64` holds.
   - For `n = 5`: `5! = 120` and `5^(5 - 1) = 5^4 = 625`. So, `120 < 625` holds.
   - The pattern suggests that this holds for all `n ≥ 3`.

2. **General Proof Idea**:
   - The proof can proceed by mathematical induction. We first check the base case `n = 3`, and then for the inductive step, we assume the statement holds for `n = k` (i.e., `k! < k^(k - 1)`) and show that it also holds for `n = k + 1` (i.e., `(k + 1)! < (k + 1)^k`).
   - The inductive step involves showing that `(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1)`. The last expression is `(k + 1)^k` because `k^(k - 1) = k^k / k` when multiplying by `(k + 1)` is not clean. Instead, we can directly use the fact that `(k + 1)! = (k + 1) * k!` and `k! < k^(k - 1)` by the inductive hypothesis, but we must also ensure `k + 1 < k + 1 + 1` is not relevant here. Wait, perhaps the best approach is to directly manipulate the inequality `(k + 1)! < (k + 1)^k`.

   However, this seems messy. A simpler way is to compare `(k + 1)!` and `(k + 1)^k` directly. For `k ≥ 3`, we have `(k + 1)! = (k + 1) * k! < (k + 1) * k^k`.

   But this is not obviously true because `k! < k^k` is equivalent to `k < k + 1`, which is trivially true.

   Alternatively, `(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1)` by IH, and `(k + 1) * k^(k - 1) = k^k + k^(k - 1)`, which is less than `(k + 1)^k = k^k + k * k^(k - 1) + ...`, but this is getting complicated.

   A much better approach is to use induction directly. For `n = 3`, we already checked this. Assume `k! < k^(k - 1)` for some `k ≥ 3`. Then:
   `(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1)`. Now, `(k + 1) * k^(k - 1) = k^k + k^(k - 1)`. We need this to be `< (k + 1)^k = k^k + k * k^(k - 1) + ...`.

   But `k^k` cancels out, and we need `k^(k - 1) < k * k^(k - 1) + ...`, i.e., `1 < k + ...`, which is true for `k ≥ 3`.

   However, the last step is not straightforward, and it might be simpler to use induction concretely.

#### Step 1: Abstract Plan

1. **Base Case (`n = 3`)**:
   - Compute `3! = 6` and `3 ^ (3 - 1) = 9`. Since `6 < 9`, the base case holds.

2. **Inductive Step (`n = k + 1`)**:
   - Assume `k! < k^(k - 1)` holds for some `k ≥ 3`.
```

```
   - We need to show `(k + 1)! < (k + 1)^k`.
   - `(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1)` by the inductive hypothesis.
   - Next, we need to compare `(k + 1) * k^(k - 1)` to `(k + 1)^k = (k + 1) * k^k / k`.
   - The inequality `k * (k + 1) ≤ k * k + k * k = 2 * k^2` for `k ≥ 1` is too loose. A better approach is as follows:
     - We want `(k + 1) * k^(k - 1) < (k + 1)^k`.
     - Divide both sides by `(k + 1)`: `k^(k - 1) < (k + 1)^(k - 1)`.
     - But:
       - This is false for `k = 3`: `3^2 = 9 < 4^2 = 16` holds. For `k = 4`: `4^3 = 64 < 5^3 = 125` holds, etc. But we need `(k + 1) * k^(k - 1) < (k + 1)^k`, not `k^(k - 1) < (k + 1)^(k - 1)`.
   - Alternatively, perhaps a direct proof is better:
     - `(k + 1) * k^(k - 1) = k^k + k^(k - 1)`.
     - We must show `k^k + k^(k - 1) < (k + 1)^k = k^k + k * k^(k - 1) + ... + 1`.
     - Since `k ≥ 3`, `k^(k - 1) < k * k^(k - 1)` and `0 < k^k` (which cancels), we get `k^(k - 1) < k * k^(k - 1)` because `1 < k` when `k ≥ 2`, so `k^k + k^(k - 1) < k^k + k * k^(k - 1) + ...` is justified.

But the above is still messy. A better way is to prove `(k + 1) * k^{k - 1} < (k + 1)^k` for `k ≥ 3`:
`(k + 1) * k^{k - 1} = k^k + k^{k - 1}`, and `(k + 1)^k = k^k + k * k^{k - 1} + ...`.
Subtract:
`(k + 1)^k - (k + 1) * k^{k - 1} = k^k + k * k^{k - 1} + ...} - (k^k + k^{k - 1}) = k * k^{k - 1} + ...` which is clearly positive because all terms are positive.
Alternatively, just note that `k^k > 0` when `k ≥ 1` is sufficient because `(k + 1)^k = k^k + ... ≥ k^k > k^{k - 1}`.

However, the direct approach is to compute `(k + 1)^k = (k + 1) * ((k + 1) * ... * 1) ≥ (k + 1) * (k * ... * 1) = (k + 1) * k^(k - 1)`, which would work except for the last factor.

Wait, we need `(k + 1)^k > (k + 1) * k^{k - 1}`. But `(k + 1)^k = (k + 1) * (k + 1)^{k - 1}`.

But `(k + 1)^{k - 1} ≥ k^{k - 1}` is not immediately obvious (and may even be false for `k = 3` (`4^2 = 16 ≥ 8 = 3^2` is true, but `4^3 = 64 ≥ 3^3 = 27` is true)).

This seems correct, but we instead keep it simple:

#### Inductive Proof Sketch:
1. Base Case (`n = 3`): `6 < 9` is true.
2. Inductive Step (`n = k + 1` assuming `k! < k^(k - 1)`):
   - We have `(k + 1)! < (k + 1) * k^(k - 1)`.
   - We need `(k + 1) * k^(k - 1) ≤ (k + 1)^k = (k + 1) * k^(k - 1)` + ...
   - Alternatively, the most straightforward induction is simply to use that `(k + 1) * k^(k - 1) < (k + 1)^k` when `k ≥ 3`.

This can be proved as follows:
`(k + 1)^k = (k + 1) * (k + 1)^{k - 1}` and `(k + 1)^{k - 1} ≥ k^{k - 1}` for `k ≥ 3` because:
`(k + 1)^2 = k^2 + 2k + 1 ≥ k^2` for all `k ≥ 1`,
`(k + 1)^3 = k^3 + 3k^2 + 3k + 1 ≥ k^3 + 3k^2 ≥ k^3 + 2k^2 ≥ k^3` for `k ≥ 2`, etc.
But we get `(k + 1)^k ≥ (k + 1) * k^{k - 1}`, which is not what we want! This is because we made a miscount. The correct relation is:
`(k + 1)^k = (k + 1) * (k + 1)^{k - 1}.`, and `(k + 1)^{k - 1} > k^{k - 1}` for `k ≥ 3` (since `l < m` implies `l^{n} < m^{n}` for `l, m > 1`). That is, `(k + 1)^k > (k + 1) * k^{k - 1} > (k + 1) * k^{k - 1} - k^{k - 1}` is not true.

Instead, `(k + 1)^k = (k + 1) * (k + 1)^{k - 1} ≥ (k + 1) * k^{k - 1}` because `(k + 1)^{k - 1} ≥ k^{k - 1}` for `k ≥ 3`.

Therefore, the full chain is:
`(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1) ≤ (k + 1)^k.`
```

```
Thus, we can prove the inductive step using the above reasoning.

#### Abstract Plan:
1. Base Case:
   - Verify the inequality for `n = 3`.
2. Inductive Step:
   - Assume the inequality holds for `n = k`, i.e., `k! < k^(k - 1)` for `k ≥ 3`.
   - Prove for `n = k + 1`, i.e., `(k + 1)! < (k + 1)^k = (k + 1) * k^(k - 1)`.
   - But it is enough to prove `(k + 1)^k ≥ (k + 1) * k^(k - 1)`:
     - Since `(k + 1)^k = (k + 1) * (k + 1)^{k - 1}`, we must show `(k + 1)^{k - 1} ≥ k^{k - 1}`.
     - This is true for `k ≥ 4` since `(k + 1)/k ≥ k/(k - 1)`.
     - Check `k = 3` separately: `4^2 = 16 ≥ 3^2 = 9` and `4^3 = 64 ≥ 3^3 = 27`, etc.

#### Lean 4 Proof Sketch with `have` Statements

```lean4
theorem induction
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by
  -- Base case: n = 3
  have h_base : (3 : ℕ)! < 3 ^ (3 - 1) := by sorry
  -- Inductive step: assume (k)! < k^(k - 1) for some k ≥ 3, prove (k + 1)! < (k + 1)^((k + 1) - 1)
  have h_inductive_step : ∀ (k : ℕ), 3 ≤ k → (k)! < k ^ (k - 1) → (k + 1)! < (k + 1) ^ ((k + 1) - 1) := by sorry
  -- Combine base case and inductive step to get the general result
  have h_general : (n)! < n ^ (n - 1) := by sorry
  sorry
```

This sets up the scaffolding of the proof using `have` statements with `sorry`.

Step-by-Step Abstract Plan

1. **Base Case (`n = 3`)**:
   - Directly check `3! = 6` and `3^(3 - 1) = 9`, hence `6 < 9`.

2. **Inductive Hypothesis (`n = k`)**:
   - Assume `k! < k^(k - 1)` for some `k ≥ 3`.

3. **Inductive Step (`n = k + 1`)**:
   - Prove `(k + 1)! < (k + 1)^((k + 1) - 1)` using:
     - `(k + 1)! = (k + 1) * k! < (k + 1) * k^(k - 1)` by IH.
     - `(k + 1)^k = (k + 1) * (k + 1)^{k - 1} ≥ (k + 1) * k^(k - 1)` if `(k + 1)^{k - 1} ≥ k^(k - 1)`. This holds for `k ≥ 3` because `(k + 1) ≥ k` and functions are increasing.

4. **Conclusion**:
   - By induction, the statement holds for all `n ≥ 3`.

Lean 4 Proof Sketch with `have` Statements and `sorry`s

```lean4
theorem induction
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by
  have h_base : (3 : ℕ)! < 3 ^ (3 - 1) := by sorry
  have h_inductive_step : ∀ (k : ℕ), 3 ≤ k → (k)! < k ^ (k - 1) → (k + 1)! < (k + 1) ^ ((k + 1) - 1) := by sorry
  have h_general : (n)! < n ^ (n - 1) := by sorry
```
```

````
  sorry
```

(Note: The 'have' statements are used to sketch the high-level structure of the proof. The actual Lean 4 proof would fill in the 'sorry' placeholders with detailed reasoning.)

### Complete Lean 4 Proof

```lean4
theorem induction
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by
  have h_base : (3 : ℕ)! < 3 ^ (3 - 1) := by
    norm_num
    <;> decide

  have h_inductive_step : ∀ (k : ℕ), 3 ≤ k → (k)! < k ^ (k - 1) → (k + 1)! < (k + 1) ^ ((k + 1) - 1) := by
    intro k hk' hk''
    have h₁ : (k + 1)! = (k + 1) * k ! := by
      simp [Nat.factorial_succ, Nat.mul_comm]
      <;> ring
      <;> simp_all [Nat.factorial]
      <;> ring
      <;> simp_all [Nat.factorial]
    rw [h₁]
    have h₂ : (k + 1 : ℕ) * k ! < (k + 1 : ℕ) * k ^ (k - 1) := by
      -- Apply the inductive hypothesis to get the inequality for 'k! < k ^ (k - 1)'
      have h₃ : k ! < k ^ (k - 1) := hk''
      have h₄ : (k + 1 : ℕ) > 0 := by linarith
      have h₅ : (k + 1 : ℕ) * k ! < (k + 1 : ℕ) * k ^ (k - 1) := by
        exact Nat.mul_lt_mul_of_pos_left h₃ (by linarith)
      exact h₅
    have h₃ : (k + 1 : ℕ) ^ ((k + 1) - 1) = (k + 1) ^ k := by
      simp [Nat.sub_add_cancel (by linarith : 1 ≤ k + 1)]
      <;> ring_nf
      <;> simp_all
    rw [h₃]
    -- Use the fact that '(k + 1) * k ^ (k - 1) < (k + 1) ^ k' to conclude the proof
    have h₄ : (k + 1 : ℕ) * k ^ (k - 1) < (k + 1) ^ k := by
      have h₅ : k ≥ 3 := by exact_mod_cast hk'
      have h₆ : (k + 1 : ℕ) * k ^ (k - 1) < (k + 1) ^ k := by
        have h₇ : (k + 1 : ℕ) ^ k = (k + 1) * (k + 1) ^ (k - 1) := by
          cases k with
          | zero => contradiction
          | succ k' =>
            cases k' with
            | zero => contradiction
            | succ k'' =>
              cases k'' with
              | zero => contradiction
              | succ k''' =>
                simp [Nat.pow_succ, Nat.mul_assoc]
                <;> ring_nf at *
                <;> omega
        rw [h₇]
        have h₈ : (k + 1 : ℕ) * k ^ (k - 1) < (k + 1) * ((k + 1) ^ (k - 1)) := by
          have h₉ : k ^ (k - 1) < (k + 1) ^ (k - 1) := by
            exact Nat.pow_lt_pow_of_lt_left (by linarith) (by
              have h₁₀ : k - 1 ≥ 2 := by
                omega
              omega)
          nlinarith
````

```
        nlinarith
      exact h₆
    nlinarith

  have h_general : (n)! < n ^ (n - 1) := by
    have h₁ : ∀ n : ℕ, 3 ≤ n → (n)! < n ^ (n - 1) := by
      intro n hn
      induction' hn with k hk IH
      · -- Base case: k = 3
        simp [Nat.factorial]
        <;> norm_num
      · -- Inductive step: assume the statement holds for k, prove for k + 1
        cases k with
        | zero => contradiction
        | succ k' =>
          cases k' with
          | zero => contradiction
          | succ k'' =>
            cases k'' with
            | zero => contradiction
            | succ k''' =>
              simp_all [Nat.factorial, Nat.pow_succ, Nat.mul_assoc]
              -- Use the inductive hypothesis and the lemma `h_inductive_step` to prove the next step
              <;>
              (try omega) <;>
              (try nlinarith) <;>
              (try
                {
                  have h₂ := h_inductive_step (k''' + 3) (by linarith) (by simpa [Nat.factorial, Nat.pow_succ, Nat.mul_assoc] using IH)
                  simp_all [Nat.factorial, Nat.pow_succ, Nat.mul_assoc]
                  <;>
                  (try omega) <;>
                  (try nlinarith)
                })
              <;>
              (try
                {
                  aesop
                })
              <;>
              (try
                {
                  nlinarith [pow_pos (by norm_num : (0 : ℕ) < 2) k''', pow_pos (by norm_num : (0 : ℕ) < 2) (k''' + 1), pow_pos (by norm_num : (0 : ℕ) < 2) (k''' + 2)]
                })
    exact h₁ n h₀

  exact h_general
```

B.4. Comparison: Direct Proof Attempt Without Lemmas

For comparison, we present below the reasoning process and generated Lean code from a direct proof attempt without using auxiliary lemmas. Among the 100 initial direct proof attempts ($N_{\text{init}} = 100$), we show the one that resulted in the fewest Lean errors.

```
### Detailed Proof and Analysis

**Problem Analysis:**
We need to prove that for all integers `n ≥ 3`, `n! < n^(n - 1)`.
```

```
5
6  **Key Observations:**
7  1. For `n ≥ 3`, `n - 1 ≥ 2` (since `3 - 1 = 2`), so `n^(n - 1)` grows faster than
       `n!` as `n` increases.
8  2. The factorial `n!` is the product of all integers from `1` to `n`, while `n^(n -
       1)` is a product of `n - 1` copies of `n` (assuming `n ≥ 3`).
9  3. The inequality `n! < n^(n - 1)` can be compared term by term, but a more precise
       estimate is needed.
10 4. For `n ≥ 3`, we have `n! < (n + 1 - 1)! = n!`? Wait, this is not helpful.
       Instead, we can compare `n!` with `n^(n - 1)`:
11    - The inequality `n! < n^(n - 1)` is equivalent to `(n - 1)! < n^(n - 2)`, but
       this is not directly useful either.
12 5. A better idea is to directly compare `n!` with `n*(n - 1)*...*(2)` and `n^(n - 1)
       = n * n * ... * n`.
13    - For `n ≥ 3`, each term `k` in the product `2 * 3 * ... * (n - 1)` is `≥ 1`
       (except `2 * 1` is skipped if `n ≥ 4`, but `n ≥ 3` is given), and each is `≤ n
       - 1` for `k ≤ n - 2`. However, comparing `k` to `n` is complicated.
14    - Alternatively, `n!` is the product of `n` factors, each `≤ n`, but `n!` is a
       product of all numbers from `1` to `n`, whereas `n^(n - 1)` is a product of `n
       - 1` copies of `n`.
15
16 But actually, induction is not straightforward here. The key is to compare `n!`
       with `n * (n - 1)^(n - 1)`, since `(n - 1)^(n - 1)` grows faster than `(n -
       1)(n - 2) ...` for `n ≥ 3`.
17
18 But `n! < n * (n - 1)!` is `n! < n * (n - 1)!`, which is true except for `n = 1`.
       So is `n * (n - 1)! < n^(n - 1)`? Not obviously, because for `n = 4`, `4 * 3! =
       24` and `4^3 = 64`, but for `n = 3`, `3! = 6` and `3^2 = 9`.
19
20 Instead, let's recall the definition of `n!` and think about how many times `n!` is
       compared to `n^(n - 1)`.
21
22 For `n ≥ 3`, each time you compare `n!` with `n * (n - 1)!`, and this is `< n * (n
       - 1)! < n * n^(n - 2)} = n^(n - 1)}` ? Seems wrong because `(n - 1)! < n^(n -
       2)}` would be true for `n ≥ 2` by induction, which is not a problem since we're
       proving it.
23
24 Perhaps a better approach is to use telescoping inequalities. Consider `n! = n * (n
       - 1) * ... * 2 * 1`, and `n^(n - 1)` is the product of `n - 1` copies of `n`.
25
26 For `n ≥ 3`, each term in the factorial is smaller than `n` (`k ≤ n - 1`).
27
28 But more precisely, `n! = n * (n - 1)!` and `(n - 1)! < n^(n - 2)` for `n ≥ 3`
       (since `n - 1 ≥ 2`).
29
30 Let's try to prove `(n - 1)! < n^(n - 2)}`. If `(n - 1)! < n^(n - 2)}` holds, then
       `n! = n * (n - 1)! < n * n^(n - 2)} = n^(n - 2)}`.
31
32 But is `n * n^(n - 2)} = n^(n - 1)}` ? Yes, because `n * n^(n - 2)} = n^(1 + n -
       2)} = n^(n - 1)}`.
33
34 Thus, `n! < n^(n - 1)}`, provided that `(n - 1)! < n^(n - 2)}`.
35
36 Wait, `n^(n - 2)}` is `n * n^(n - 3)}` if `n ≥ 3`.
37
38 So the induction is:
39 - Base case `n = 3`: `2! = 2` and `3^2 = 9`, `2 < 9` holds.
40 - Inductive step: Assume for `n`, `(n - 1)! < n^(n - 2)}`. Then `n * (n - 1)! < n *
       n^(n - 2)} = n^(n - 1)}`? But `n * n^(n - 2)} = n^(n - 2 + 1)} = n^(n - 1)}`.
       So `n! = n * (n - 1)! < n^(n - 1)}`.
41
42 This is a valid induction, provided that `(n - 1)! < n^(n - 2)}` is true for `n ≥
       3`.
43
44 Now, recall that for `n ≥ 3`:
```

```
`n! = n * (n - 1)!` and `(n - 1)! < n^(n - 2)` if `n ≥ 3`.

But is `(n - 1)! < n^(n - 2)}` true for `n ≥ 3`?

For `n = 3`, LHS = 2, RHS = 1. False. Hmm, `3! = 6` and `3^(3 - 3)` = 1? No, `3 ≥
    3` and `3^(3 - 2)} = 3^{1}} = 3`, `2 < 3`, so true.

Wait no. For `n = 4`, `(3)! = 6`, and `4^{4 - 2}} = 4^{2}} = 16`, `6 < 16` true.

For `n = 5`, `(4)! = 24`, `5^{5 - 2}} = 5^3 = 125`, `24 < 125` true.

Hypothesis that `(n - 1)! < n^(n - 2)}` for all `n ≥ 3`.

Base case `n = 3`, `2! = 2 < 3^{1} = 3` true.

Inductive step: Assume `(n - 1)! < n^(n - 2)}` for `n ≥ 4`.

We need `n! < (n + 1)^(n - 1)}`:
`n! = n * (n - 1)! < n * n^(n - 2)} = n^(n - 1)}`, since `(n - 1)! < n^(n - 2)}`
    holds by the inductive hypothesis for `n ≥ 3`.

This works!

**Conclusion:** The inequality `n! < n^{n - 1}` holds for all `n ≥ 3` (`n >= 3`).

But wait, the original Lean statement is `n : ℕ` and `3 ≤ n`, so indices are clear.

We can proceed directly using induction. The `have` statements will reflect the
    proof.

**Lean 4 Proof Sketch:**

1. **Base Case (`n = 3`)**: Check that `3! < 3^{3 - 1}` or `6 < 9`.
2. **Inductive Step**: Assume `(k)! < k^{k - 1}` for some `k ≥ 3`, and prove `(k +
    1)! < (k + 1)^{k}` using the assumption `k! < k^{k - 1}`.

### Abstract Step-by-Step Plan

1. **Base Case (`n = 3`)**:
   - Compute `3! = 6`.
   - Compute `3^{3 - 1} = 3^2 = 9`.
   - Since `6 < 9`, the statement holds.

2. **Inductive Step (`n → n + 1`)**:
   - Assume `n! < n^{n - 1}` for some `n ≥ 3` (induction hypothesis).
   - Prove `(n + 1)! < (n + 1)^n`:
     - `(n + 1)! = (n + 1) * n!` (by definition).
     - By hypothesis, `n! < n^{n - 1}`.
     - Thus, `(n + 1)! < (n + 1) * n^{n - 1}`.
     - It remains to prove `(n + 1) * n^{n - 1} ≤ (n + 1)^n`, i.e., `n^{n - 1} ≤
    (n + 1)^{n - 1}` (if `n + 1 ≥ 2`? Always true since `n ≥ 2`).
      * Since `n ≥ 2`, `n^{n - 1} ≤ (n + 1)^{n - 1}` because `n ≤ n + 1` and the
    exponent is `n - 1 ≥ 2`, making the term `(n + 1)^{n - 1}` strictly greater
    than `n^{n - 1}}`.

### Lean 4 `have` Statements

```lean4
theorem induction
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by
  have h_main : (n)! < n^(n - 1) := by
    sorry
  sorry
```
```

````
```

### Explanation of `have` Statements:
1. The `have` statement `h_main` directly encodes our goal `(n)! < n^(n - 1)`.
   Inside the proof, we would perform induction, but in a structured Lean proof,
   we break it into a single `have h_main`.
  - The actual proof would expand on `h_main` by leveraging induction, but for this
   sketch, it suffices to rely on the good old induction hypothesis.

### Complete Lean 4 Proof

```lean4
theorem induction
  (n : ℕ)
  (h₀ : 3 ≤ n) :
  (n)! < n^(n - 1) := by
  have h_main : (n)! < n^(n - 1) := by
    have h₁ : ∀ n : ℕ, 3 ≤ n → (n)! < n ^ (n - 1) := by
      intro n hn
      induction' hn with n hn IH
      · -- Base case: n = 3
        norm_num [Nat.factorial]
      · -- Inductive step: assume the statement holds for n, prove for n + 1
        cases n with
        | zero => contradiction -- n cannot be zero since 3 ≤ n
        | succ n =>
          cases n with
          | zero => contradiction -- n cannot be one since 3 ≤ n
          | succ n =>
            cases n with
            | zero => contradiction -- n cannot be two since 3 ≤ n
            | succ n =>
              simp_all [Nat.factorial, Nat.pow_succ, Nat.mul_comm, Nat.mul_assoc,
   Nat.mul_left_comm]
              <;>
                (try omega) <;>
                (try
                  {
                    nlinarith [pow_pos (by decide : (0 : ℕ) < 2) n, pow_pos (by
   decide : (0 : ℕ) < 2) (n - 1)]
                  }) <;>
                (try
                  {
                    ring_nf at *
                    <;>
                      nlinarith [pow_pos (by decide : (0 : ℕ) < 2) n, pow_pos (by
   decide : (0 : ℕ) < 2) (n - 1)]
                  }) <;>
                (try
                  {
                    simp_all [Nat.factorial_succ, Nat.mul_comm, Nat.mul_assoc,
   Nat.mul_left_comm]
                    <;>
                      norm_num at *
                    <;>
                      ring_nf at *
                    <;>
                      nlinarith [pow_pos (by decide : (0 : ℕ) < 2) n, pow_pos (by
   decide : (0 : ℕ) < 2) (n - 1)]
                  })
    exact h₁ n h₀
  exact h_main
```
````
````