

# MA-LoT: Multi-Agent Lean-based Long Chain-of-Thought Reasoning enhances Formal Theorem Proving

Ruida Wang<sup>\*1</sup> Rui Pan<sup>\*2</sup> Yuxin Li<sup>\*3</sup> Jipeng Zhang<sup>3</sup> Yizhen Jia<sup>2</sup>  
Shizhe Diao<sup>4</sup> Renjie Pi<sup>3</sup> Junjie Hu<sup>1</sup> Tong Zhang<sup>2</sup>

## Abstract

Solving mathematical problems using computer-verifiable languages like Lean has significantly impacted mathematical and computer science communities. State-of-the-art methods utilize single Large Language Models (LLMs) as agents or provers to either generate complete proof or perform tree searches. However, single-agent methods inherently lack a structured way to combine high-level reasoning in Natural Language (NL) with Formal Language (FL) verification feedback. To solve these issues, we propose **MA-LoT: Multi-Agent Lean-based Long Chain-of-Thought** framework, (to the best of our knowledge), the first multi-agent framework for Lean4 theorem proving that balance high-level NL reasoning and FL verification in Long CoT. Using this structured interaction, our approach enables deeper insights and long-term coherence in proof generation, with which past methods struggle. We do this by leveraging emergent formal reasoning ability in Long CoT using our novel *LoT-Transfer Learning* training-inference pipeline. Extensive experiment shows that our framework achieves **54.51%** accuracy rate on the Lean4 version of MiniF2F-Test dataset, largely outperforming GPT-4 (22.95%), single-agent tree search (InternLM-Step-Prover, 50.70%), and whole-proof generation (DeepSeek-Prover-v1.5, 48.36%) baselines. Furthermore, our findings highlight the potential of combining Long CoT with formal verification for a more insightful generation in a broader perspective.

<sup>\*</sup>First Authors <sup>1</sup>Department of Computer Science, University of Wisconsin-Madison <sup>2</sup>Department of Computer Science, University of Illinois Urbana-Champaign <sup>3</sup>Hong Kong University of Science and Technology <sup>4</sup>NVIDIA. Correspondence to: Ruida Wang <rickyskywalker@cs.wisc.edu>.

## 1. Introduction

Formal reasoning is a cornerstone of human intelligence and a key objective in machine learning (Newell & Simon, 1956), often evaluated through rigorous mathematical derivations (Yang et al., 2024a). With the rise of Large Language Models (LLMs), Chain-of-Thought (CoT) prompting has emerged as a method to formalize reasoning by generating intermediate steps. This approach not only improves interpretability but also enhances reasoning performance (Wei et al., 2022).

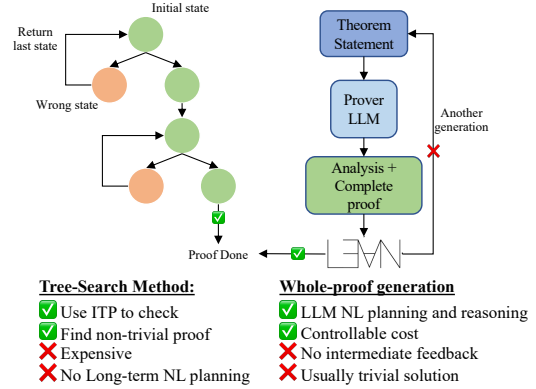
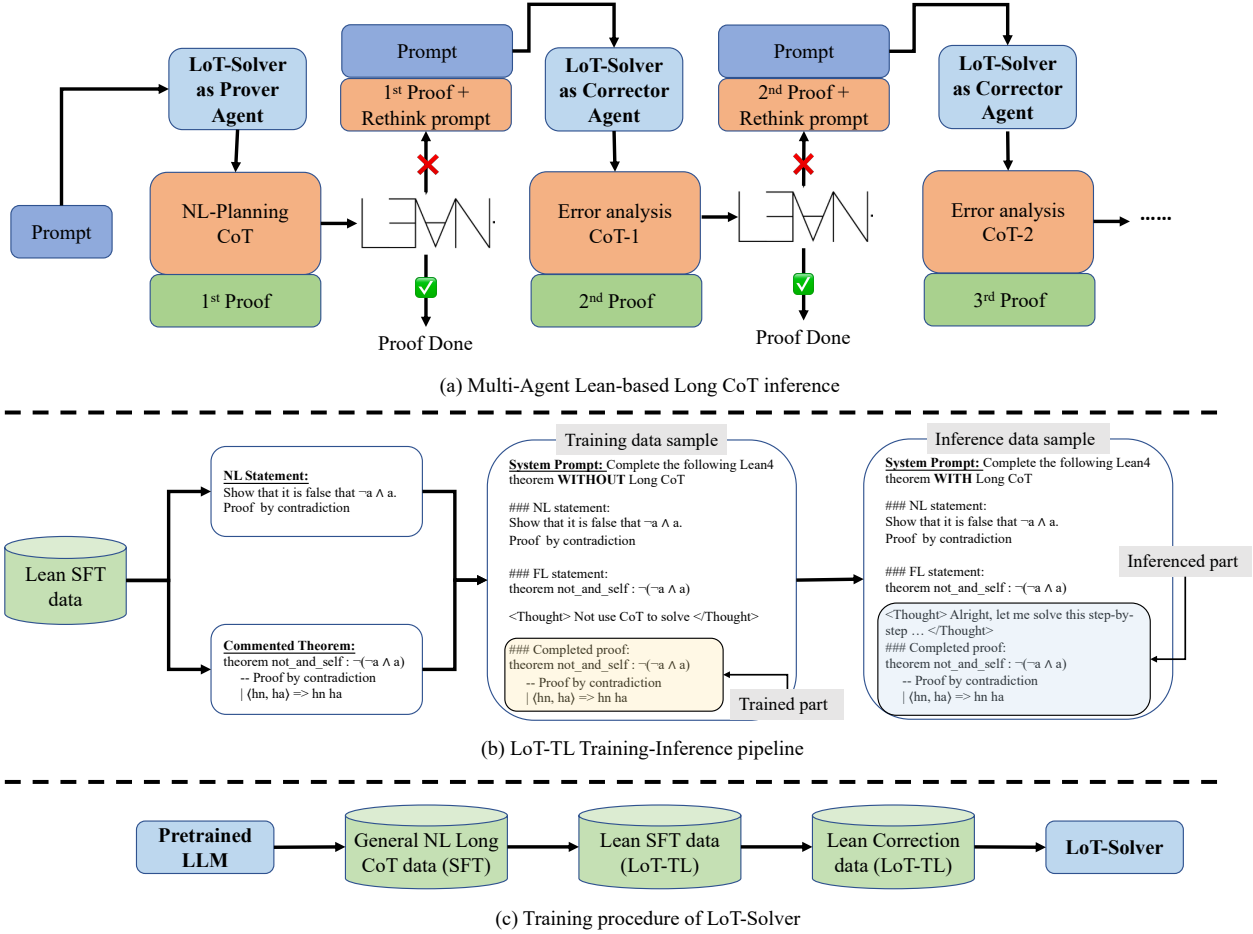


Figure 1. Two main directions of FL theorem proving using LLMs: Single agent tree-search and whole-proof generation with their advantages/disadvantages.

However, the ambiguity of Natural Language (NL) complicates the verification of intermediate steps, particularly in advanced mathematics, where theorem proving is prevalent. This challenge is exacerbated by the growing complexity of modern mathematics, which makes proof verification highly demanding and can lead to errors, as seen in the prolonged validation of Fermat’s Last Theorem (Wang et al., 2024). To address this, researchers propose grounding reasoning in first-order logic, enabling automated verification via Formal Language (FL) verifiers. This framework ensures rigor and has led to the development of tools like Lean (De Moura et al., 2015; Moura & Ullrich, 2021), Isabelle (Paulson, 1994), and HOL Light (Harrison, 2009) for verifiable theorem proving.

However, writing mathematical proofs in FL requires signif-



**Figure 2. MA-LoT Framework:** (a) Multi-agent Lean4 theorem proving framework: The *LoT-Solver* model functions as the prover agent to generate initial Lean4 proofs with emergent NL planning for Lean proof in Long CoT (orange block); then it acts as corrector agent to analyze error from Lean executor in Long CoT to output a refined proof. (b) LoT-Transfer Learning (TL): The novel training-inference pipeline makes formal reasoning ability to emerge in Long CoT (L-CoT) without the need for specifically annotated data. This is achieved by adjusting the system prompt to control the on/off of L-CoT in training and inference. (c) Training procedure of *LoT-Solver*: We use normal SFT to train general NL L-CoT, use *LoT-TL* to train on Lean SFT and correction data to make Lean L-CoT emergent capability in LLM.

icant expertise and effort, as most proofs involve extensive repetition and application of low-resource functions (Jiang et al., 2022). With the rapid progress of LLMs, research has explored LLMs’ application in FL reasoning to automate theorem proving (Polu & Sutskever, 2020; Polu et al., 2022; Jiang et al., 2021; 2022; Yang et al., 2024b; Xin et al., 2024b; Wang et al., 2024; Wu et al., 2024a; Kumarappan et al., 2024; Lin et al., 2024). Prior research follows two main approaches, namely, tree-search (Jiang et al., 2021; 2022; Lin et al., 2024; Xin et al., 2024b; Wang et al., 2024) and whole-proof generation (Polu & Sutskever, 2020; Polu et al., 2022; Yang et al., 2024b; Wu et al., 2024a; Kumarappan et al., 2024). The summary of dis/advantages of these two methods can be found in Figure 1

Tree-search methods train an LLM agent to iteratively generate proof steps by predicting the next tactic based on the current proof state. This is achieved through either direct

code writing (Polu & Sutskever, 2020; Polu et al., 2022; Xin et al., 2024b; Wu et al., 2024a; Lin et al., 2024) or retrieval-based techniques (Yang et al., 2024b; Kumarappan et al., 2024). These methods apply FL executor to verify after each step of generation and is able to discover some non-trivial proofs. However, as proof complexity increases, tree-search methods become computationally expensive and lack high-level NL planning to control the overall structure of the proof. This causes the tree-search method unable to find some structured proof that requires high-level analysis of the natural language meaning of the question.

In contrast, whole-proof generation treats theorem proving like the code generation problem, where LLMs generate the entire proof in a single attempt using either supervised training (Wang et al., 2024; Xin et al., 2024b) or prompt engineering (Jiang et al., 2021; 2022). This approach leverages the NL reasoning and high-level planning capabilities

of LLMs with predictable computation costs, but it lacks intermediate feedback from FL executors. As a result, the generated proofs often lack post-hoc analysis of errors and tend to perform badly on tedious questions that require non-trivial solutions. In summary, existing single agent (or model) approaches struggle to balance the NL reasoning with the verifiability constraints of FL, motivating the need of a more comprehensive framework.

To address the above challenges, we introduce **MA-LoT**: *Multi-Agent Lean-based Long Chain-of-Thought* framework, (to the best of our knowledge), the first multi-agent framework for Lean4 theorem proving. The multi-agent inference method for **MA-LoT** is shown in Figure 2 (a). **MA-LoT** framework can find both well-structured and non-trivial proofs by employing a collaborative agent framework and the emergent formal thinking capability in Long CoT. Specifically, the multi-agent framework contains a prover agent to write well-structured proof and a corrector agent to analyze error messages from the Lean executor. The Long Chain-of-Thought (CoT) guides the model to think thoroughly before making outputs. We achieve both agent-type control and high-level NL planning in Long CoT. Additionally, we also improve the self-reflection capability by integrating Lean verification results in Long CoT. The two systems work synergically to achieve higher formal reasoning ability through comprehensive proof planning and systematic error analysis.

To support the **MA-LoT** framework we designed above, we develop the novel *LoT-Transfer Learning* (TL) training-inference pipeline to train the *LoT-Solver* model as shown in Figure 2 (b) & (c). Through this pipeline, formal reasoning ability emerges in Long CoT without needing specifically annotated data. It is done by leveraging transfer learning to integrate capabilities in NL Long CoT reasoning, theorem proof SFT data, and correction data together. The structured adaptation enables the model to be aware of formal states and tactics while maintaining strong NL planning capabilities, leading to more coherent and insightful formal proof generation.

Extensive experiments demonstrate that **MA-LoT** framework effectively enhances the model’s formal reasoning ability through multi-agent design and emergent formal reasoning in Long CoT. The framework can successfully proof some of the advanced IMO, and AIME problems in the MiniF2F dataset (Zheng et al., 2021), with which existing models struggle. Under similar sampling budgets, our framework achieves a 54.51% accuracy rate, surpassing state-of-the-art whole-proof generation models (DeepSeek-Prover-V1.5(Xin et al., 2024b), 48.36%) and tree-search baseline (InternLM-Step-Prover(Wu et al., 2024a), 50.70%).

We summarize our contributions as follows: (1) We introduce **MA-LoT** (to the best of our knowledge) the first

multi-agent framework to balance NL reasoning and FL verification under the Long CoT paradigm for Lean4 theorem proving. (2) We propose the method of using Long CoT to synergically combine the nature of NL and FL, allowing the model to generate in-depth and insightful formal theorem proofs through NL planning and analysis. (3) We develop *LoT-TL*, a training-inference pipeline that makes field-specific Long CoT capabilities emerge to LLMs without requiring explicitly annotated datasets.

Our framework has broad potential beyond Lean4 theorem proving, demonstrating how formal verification can be effectively integrated with Long CoT reasoning. This approach demonstrates the potential for structured, reflective, and adaptable general text generation through iterative planning and error analysis on formal executors. To accelerate advancements of this field, we plan to open-source our code, dataset, and models in the near future.

## 2. Methodology

In this section, we detail the development of the **MA-LoT** framework and training procedure of *LoT-Solver* model for Lean4 theorem proving. Our framework is designed to make formal reasoning ability to emerge in Long CoT to achieve deep integration between Natural Language (NL) and Formal Language (FL). This is done under an extreme scarcity of NL-FL aligned data faced by the entire field (Wang et al., 2024). We first outline the preliminaries of LLM formal theorem proving in Section 2.1. Then, we describe the *LoT-Transfer Learning* (LoT-TL) training pipeline in Section 2.2. Finally, we present comprehensive details on how our trained model facilitates **MA-LoT** framework for Lean4 proof writing in Section 2.3.

### 2.1. Preliminaries

We introduce some preliminary knowledge of applying multi-agent Long Chain-of-Thought (CoT) LLMs to Lean4 formal theorem proving.

Current state-of-the-art methods treat Lean4 code as plain text and input them to LLMs. Some works (Yang et al., 2024b; Wu et al., 2024a) apply LLMs as agents to perform tree-search based on the current proof state. They convert theorem statements and proof states (including premises and goals) into text input for LLMs and ask it to generate the possible next tactic. Other works (Xin et al., 2024b; Wang et al., 2024) treat theorem proving like a code-generation task and develop whole-proof generation methods. This method provides the LLMs with NL instruction, NL theorem statement, and Lean4 formal statement; the intended outcome is the complete Lean4 proof in a single pass, which harnesses the NL reasoning capabilities of LLMs by letting it write the NL plans during proving. The input-output ex-

amples for both tree-search and whole-proof generation are presented in Appendix B.

The Long CoT LLMs, represented by O1 (OpenAI, 2024) make long internal NL thinking before outputting the final answer. It largely enhances the NL math reasoning ability of LLMs through self-reflection and correction in Long CoT. However, it still struggles to provide rigorous NL proofs and typically has relatively low FL capability.

Our approach integrates the strengths of both tree-search and whole-proof generation by employing a multi-agent system. We apply Long CoT to regulate the interaction between NL reasoning in LLMs and Lean4 verifier feedback, enabling the model to provide more structured and insightful proofs.

## 2.2. LoT-TL Training Pipeline

This section introduces a simple but highly effective training pipeline-*LoT-Transfer Learning* (TL), which makes Lean4 field-specific Long CoT ability emerge in LLMs without the need for Lean4 Long CoT data. This strategy leverages system prompts to regulate training and inference behaviors; it can be divided into three stages: (1) collecting field-specific Supervised Fine-Tuning (SFT) data (Section 2.2.1), (2) training the model on general natural language Long CoT tasks (Section 2.2.2), and (3) training the model using the transfer learning method on SFT and correction data to make formal Long CoT ability emerge (Section 2.2.3). We use DeepSeek-Prover-v1.5-SFT (Xin et al., 2024b) as the base model. Although we focus on Lean4, our framework shows potential to extend to applying Long CoT reasoning in general fields without requirement for RL or special data annotation.

### 2.2.1. OBTAIN SFT DATA

The first step of *LoT-TL* pipeline is to gather a moderate amount of NL-FL aligned SFT data for the specific target field (in our case, Lean4). However, existing open-source datasets do not meet the requirement. They are typically small in size (e.g., MiniF2F (Zheng et al., 2021)), or omit NL annotations (e.g., DeepSeek-Prover-v1 dataset (Xin et al., 2024a)), or exhibit relatively low NL quality (e.g., OBT (Wang et al., 2024)), or lack of Lean4 proofs (e.g., Lean-Workbook (Ying et al., 2024)).

To address this, we compile a new Lean theorem proof dataset named LoT-ProveData (LoT-PD), containing 54,465 data records. Each record contains Lean4 theorem statements, verified proofs with NL explanations as comments, and NL statements. The Lean4 theorem proofs come from two sources: the DeepSeek-Prover-v1 dataset and the annotated Lean-Workbook using TheoremLlama and DeepSeek-Prover-v1.5-RL. Next, inspired by the analysis-then-generate approach in Wang et al. (2023), we employ

Qwen-2.5-72B to provide an analysis of Lean4 proofs, followed by writing NL proof based on the analysis. Finally, we integrate these NL proofs as comments in the Lean4 code by Qwen. For data records lacking NL statements, we generate NL statements using a similar method. The core components of our LoT-ProveData is:

```
{FL Statement, Commented FL proof, NL
→ statement}
```

During proof generation for the ProverData, some incorrect proofs were also produced. We recorded these alongside their error messages to form LoT-CorrectionData (LoT-CD), consisting of 64,912 records of correct-incorrect Lean4 proofs plus associated error messages and NL statement and proof. LoT-CD is used to train the model’s error analysis and correction capability. The core part of the LoT-CorrectionData

```
{FL Statement, Correct FL proof, Error
→ messages, Incorrect FL proof, NL
→ statement}
```

These datasets work together to improve both the prover and corrector agent’s capability, acting as the source of strong NL-FL joint thinking ability for our multi-agent framework.

### 2.2.2. NL LONG CoT TRAINING

In the second stage, we train a normal instruction-finetuned model to acquire Long CoT reasoning for general NL tasks. We use the OpenO1-SFT-Pro dataset provided by Open-Source-O1 (2024), a 126k records dataset for general NL question-answering on math and science topics with Long CoT for training. We apply standard next-token prediction SFT, guiding the model to produce Long CoT before it outputs final answers. Throughout the NL Long CoT training, we set the system prompt as follows to explicitly instruct the model to use the Long CoT approach:

```
You are a helpful assistant who will
→ solve every problem **WITH** Long
→ Chain-of-Thought
```

This prompt effectively “switches on” Long CoT reasoning. The training input includes the system prompt and NL question, with the expected output being the Long CoT and final answer. After training, we observe that the model gains robust NL Long CoT capabilities, which serve as a base for agents to analyze and interact with Lean. However, when applied to Lean4 reasoning, it tends to provide only NL solutions rather than outputting Lean4 code in its output section, indicating the need for further alignment.

### 2.2.3. FIELD-SPECIFIC ALIGNMENT

In the final stage of the training process of *LoT-TL* pipeline, we train the model to make Lean4 Long CoT ability emerge. This is achieved in the training stage by switching to a different system prompt that indicates the model not to



use Long CoT and to use a placeholder for the Long CoT content to keep the Long CoT backbone, while not requiring providing any actual Long CoT data in Lean4. Specifically, the system prompt is:

You are a helpful assistant who will  
 → solve every problem **\*\*WITHOUT\*\***  
 → Long Chain-of-Thought

and the placeholder Long CoT is:

The user asks not to solve with Long  
 → CoT, so I will directly write the  
 → answer.

Using this setup, we first train on the LoT-ProveData for formal theorem proving ability, then train on the LoT-CorrectionData to make the model learn error-analysis and correction skills. The example of training data can be found in Appendix E. We also adopt the curriculum learning data sorting method from Wang et al. (2024) to stabilize training. After training, we find that the Long CoT Lean4 proving and error analysis ability emerges in the LLMs when using the system prompt to turn on Long CoT in inference. We conclude the effectiveness of the TL framework because it preserves the structure of Long CoT and enables the model to activate such capability when instructed.

Following these training steps, we obtain *LoT-Solver*, which has the emergent Lean reasoning capability in Long CoT from data without Lean Long CoT annotation. This makes it able to perform the roles of both whole-proof prover and tree-search corrector with deep NL-FL joint thinking.

### 2.3. Multi-Agent FL Proof Writing

This section presents (to the best of our knowledge) the first multi-agent framework that combines the advantages of whole-proof generation and tree-search methods under the Lean-based Long CoT paradigm. We use the *LoT-Solver* as the base model for both agents. Under this setup, we use the prover agent to write a complete proof draft (Section 2.3.1) and apply the corrector agent to analyze and correct the proof based on Lean verifier feedback (Section 2.3.2).

#### 2.3.1. PROVER AGENT

The Prover agent writes the initial Lean4 proof using a whole-proof generation strategy, then submits it to the Lean4 verifier to check the correctness and passes it to the corrector agent if the proof is wrong. We use the system prompt to turn on the Long CoT reasoning and use a specific header in Long CoT to guide the model to make a high-level proof plan. Here is the instruction template for the prover agent:

```
{... **WITH** Long CoT ...}  
### Instruction:  
{NL statement}  
{FL statement}  
### Response:
```

Alright, I should do the following:

1. Provide the natural language  
 → analysis for the theorem based on  
 → the Natural language theorem  
 → statement.
2. Draft the Lean4 tactics I should use  
 → to solve the problem
3. Write the output Lean4 code.

The complete example of prover agent input and output can be found in Appendix F. The emergent Lean reasoning ability in the Long CoT enables the model can write a better-structured proof based on its high-level plan compared to direct proof generation. Upon generating the proof, the prover agent submits it to the Lean evaluator for verification. If incorrect, the theorem is passed to the corrector agent for further refinement.

#### 2.3.2. CORRECTOR AGENT

The corrector agent functions like the tree-search method. Upon receiving a wrong proof and Lean verifier feedback, it systematically analyzes them in Long CoT. Then, after re-evaluating the proof strategy and rethinking, the model generates a revised proof that intends to solve the errors.

The instruction prompt remains identical to the prover agent. We incorporate the incorrect proof and feedback from the Lean verifier in the Long CoT followed by instructions to direct the model to analyze the error and formulate a revised proof. Detailed examples of such prompts are available in Appendix F. Then, the corrector agent will pass the new proof to the Lean4 verifier, if the new proof is still wrong, iteratively analyze the errors until success or reach the max retry limit.

The corrector agent enhances theorem proving by enabling deeper reflection and systematic exploration of alternative proof strategies based on error messages. This iterative correction process increases the likelihood of discovering non-trivial proofs while maintaining computational efficiency.

## 3. Experiments

We conduct comprehensive experiments on the MiniF2F-Lean4 (Zheng et al., 2021) dataset to assess the performance of **MA-LoT** framework in formal proof writing. Specifically, we evaluate the superiority of our multi-agent Long CoT system in writing better-structured and more insightful proofs by additional proof it could find under similar sampling budget (Section 3.3). Additionally, we perform studies on our corrector agent (Section 3.4), training components of *LoT-Solver* (Section 3.5), and case study (Section 3.6) to further analyze the impact of individual components.

Method	Model size	Sample budget	MiniF2F-Valid	MiniF2F-Test	Average
<i>Tree-search Methods</i>					
<b>ReProver (Yang et al., 2024b)</b>	229M	-	-	26.5%	-
<b>Llemma (Azerbayev et al., 2023)</b>	34B	$1 \times 32 \times 100$	27.9%	25.8%	26.85%
<b>Expert Iteration (Polu et al., 2022)</b>	837M	$8 \times 8 \times 512$	41.2%	36.6%	38.9%
<b>Lean-STaR (Lin et al., 2024)</b>	7B	$64 \times 1 \times 50$	-	46.3%	-
<b>InternLM2.5-StepProver (Wu et al., 2024a)</b>	7B	$2 \times 32 \times 600$	56.0%	50.7%	53.35%
<i>Whole-proof generation</i>					
<b>GPT-4-Turbo (Achiam et al., 2023)</b>	> 1T		25.41%	22.95%	24.18%
<b>DeepSeek-Math (Shao et al., 2024)</b>	7B		25.80%	24.60%	25.20%
<b>Gemini-1.5-pro (Reid et al., 2024)</b>	-	pass@128	29.92%	27.87%	28.90%
<b>TheoremLlama (Wang et al., 2024)</b>	8B		38.52%	35.66%	37.66%
<b>DeepSeek-Prover-v1.5-RL (Xin et al., 2024b)</b>	7B		54.10%	48.36%	51.23%
<b>LoT (whole-proof)</b>		pass@128	62.70%	52.05%	57.42%
<b>MA-LoT</b>	7B	$64 + 32 \times 2$	<b>64.34%</b>	<b>54.51%</b>	<b>59.22%</b>
<b>MA-LoT</b>		cumulative	<b>65.98%</b>	<b>56.97%</b>	<b>61.48%</b>

Table 1. Main experimental results of **MA-LoT**. **LoT** (whole-proof) indicates the whole-proof generation results of our *LoT-Solver* model, **MA-LoT** with sampling budget  $64 \times 2$  is the major result of our **MA-LoT**. For baselines using tree-search baseline, we take the closest available sample budget result, for baselines using whole-proof generation, we take the pass@128 sample budget.

### 3.1. Experiment Setup

#### 3.1.1. DATASET AND TASK

In this paper, we assess **MA-LoT**’s Lean4 reasoning capabilities on the MiniF2F-Test and Valid<sup>1</sup> datasets (Zheng et al., 2021; Yang et al., 2024b; Wang et al., 2024). MiniF2F is a widely used and challenging benchmark for formal theorem proving (Frieder et al., 2024), which is adopted in nearly all major studies in the field (Jiang et al., 2021; Polu et al., 2022; Jiang et al., 2022; Wu et al., 2024a; Lin et al., 2024; Yang et al., 2024b; Xin et al., 2024b; Wang et al., 2024; Azerbayev et al., 2023).

Both the test and validation datasets contain 244 Lean4 statements. The range of problems varies from high-school competition questions to undergraduate-level theorem proofs. It contains 488 problems from three sources: (1) 260 problems sampled from the MATH dataset (Hendrycks et al., 2021); (2) 160 problems from high-school math competitions including AMC, AIME, and AMO; (3) 68 manually crafted problems at the same difficulty level as (2). Our task is to query the LLM to generate Lean4 proofs for MiniF2F problems based on their formal statements and NL descriptions. To minimize computational overhead, imports are manually configured.

#### 3.1.2. BASELINES

To highlight **MA-LoT**’s capabilities, we select some of the most competitive baselines in recent years, cov-

ering both tree-search and whole-proof generation approaches. For tree-search methods, we include: Expert Iteration (Polu et al., 2022), Llemma (Azerbayev et al., 2023), ReProver (Yang et al., 2024b), Lean-STaR (Lin et al., 2024), and InternLM2.5-StepProver (Wu et al., 2024a) as our baselines. For whole-proof generation baselines, we include closed-source LLMs, represented by GPT-4-Turbo (Achiam et al., 2023) and Gemini-1.5 (Reid et al., 2024), and open-source expert models, such as DeepSeek-Math (Shao et al., 2024), TheoremLlama (Wang et al., 2024), and DeepSeek-Prover-v1.5-RL (Xin et al., 2024b).

For whole-proof generation baselines, we set the sample budget to pass@128 with 4096 context length, balancing robustness and manageable GPU consumption. For tree-search methods, we align the search cost as closely as possible to whole-proof generation.<sup>2</sup>

### 3.2. Implementation Details

In the training process of the model, we use OpenAI-SFT-Pro, LoT-ProverData, and LoT-CorrectionData to train the base model DeepSeek-Prover-v1.5-SFT. For different training stages, the learning rate is as follows: 1E-5 for NL Long CoT training, 1E-7 for LoT-TL on LoT-PD, and 1E-6 for LoT-CD. The total computational cost for training is around 1 GPU day, and evaluation is 11 GPU days on a  $4 \times$  H100-96G cluster<sup>3</sup>. To evaluate our framework in detail, we present three sets of results, including: (1) **LoT**

<sup>1</sup>Although DeepSeek-Prover-v1.5 declared that they applied MiniF2F-Valid for training, yet its performance does significantly differentiate from other methods. Thus, we still keep this baseline.

<sup>2</sup>This explains why we exclude RMaxTS for DeepSeek-Prover-v1.5, as its smallest disclosed sample budget ( $1 \times 3200$ ) not a comparable result to our baselines.

<sup>3</sup>The high evaluation cost due to accelerate inference methods like vLLM unable to fit our machine

Method	Prover	1 round	2 round	3 round
MiniF2F-Test	51.64%	53.28%	54.51%	55.33%

Table 2. Result of different rounds of corrector agent correction

(whole-proof): pass@128 whole-proof generation result of *LoT-Solver*. (2) **MA-LoT**: Our primary evaluation result, where the prover performs 64 whole-proof generations and undergoes two rounds of corrector refinement<sup>4</sup>. (3) Cumulative Results: A combined evaluation aggregating all **LoT** models outputs obtained throughout the experiment process.

Our code, model, and data will be published in <https://github.com/RickySkywalker/LeanOfThought-Official>

### 3.3. Results

Table 1 presents our main results, showing that **MA-LoT** achieves **54.51%** accuracy rate on MiniF2F-Test benchmark, and 52.05% for *LoT-Solver* using whole proof-completion. Detailed analysis also shows that our models can solve some IMO and AIME problems that previous models struggled with. Both models surpass state-of-the-art tree-search (InternLM2.5) and whole-proof generation (DeepSeek-Prover-v1.5) baselines, demonstrating that our proposed multi-agent framework based on Long CoT excels in formal theorem proving.

**MA-LoT** outperforms all tree-search baselines by at least 3.81% because its prover agent constructs proofs with high-level NL planning using emergent Lean Long CoT reasoning capability. This indicates our prover agent can leverage LLMs’ strong NL reasoning ability, which leads to more comprehensive proofs. Additionally, **MA-LoT** surpasses whole-proof generation baselines by at least 6.15%, as its corrector agent analyzes, reflects, and reformulates proofs based on Lean4 executor feedback in Long CoT. The strong performance also demonstrates the effectiveness of our ideas of integrating FL verification in NL Long CoT reasoning with its emergent capability. Notably, although **MA-LoT** is based on DeepSeek-Prover-v1.5-SFT, it outperforms its RL-trained variant by 6.15%. This suggests that our multi-agent training framework and Lean-based Long CoT methodology align more naturally with formal theorem proving than RL alone.

The comparison between **LoT** (whole-proof) and **MA-LoT** further highlights the importance of our multi-agent framework. We observe a 2.46% improvement by reallocating the computation resources from the prover agent doing more whole-proof generation to the corrector agent to analyze and refine proof based on Lean executor feedback. This validates

<sup>4</sup>Because we don’t need to pass correct proof to corrector, two rounds of correction is approximately the same as one round of whole-proof generation

Method	MiniF2F-Test
DeepSeek-Prover-v1.5-SFT (base model)	46.31%
LoT-Solver witch-off Long CoT	49.18%
w/o Long CoT training (on RL model)	48.36%
base model + Long CoT	46.72%
base model + Long CoT + SFT	50.00%
<b>LoT-Solver</b>	<b>51.64%</b>

Table 3. Ablation study result in pass@64

the necessity of an iterative prover-corrector multi-agent system over relying solely on a prover agent. In summary, these results confirm that Long CoT reasoning, combined with formal verification and a multi-agent paradigm, enhances the discovery of non-trivial and in-depth proofs, thereby validating the effectiveness of our proposed method.

### 3.4. Corrector Agent study

To assess the impact of the corrector agent in our multi-agent system, we present the cumulative accuracy on MiniF2F across different rounds in Table 2. The prover column represents the pass@64 accuracy rate of the prover agent in whole-proof generation, while Round-*i* columns indicate successive correction rounds, each permitting up to 64 refinements per theorem. The results indicate during the first three correction rounds, the corrector agent successfully refines an average of 1.12% of theorems. Our analysis shows that most corrected proofs belong to IMO, AIME, and high-level MATH problems, which are particularly difficult for prior models. This highlights the corrector agent’s ability to analyze feedback from the Lean4 executor feedback using emergent Lean capability in Long CoT to discover non-trivial proofs. The case study for the analysis and re-generation of new proofs can be found in Section 3.6 and Appendix D.

### 3.5. Ablation Study

To evaluate the effectiveness of each component of our *LoT-TL* training pipeline, we conduct this thorough ablation study. We demonstrate that the elements in *LoT-TL* pipeline work synergistically to strengthen the model’s formal theorem-proving capability through the integration of FL in Long CoT. We apply the pass@64 accuracy rate on the whole-proof generation method for this set of experiments. The results are presented in Table 3.

#### 3.5.1. EFFECT OF TRAINING STAGES

We evaluate training progression by measuring performance across key intermediate models, namely *base model*, *base model + Long CoT*, and *base model + Long CoT + SFT*, as shown in Table 3. Results indicate that training solely on NL CoT data provides minimal improvement, suggesting that NL CoT reasoning alone does not make Lean CoT

capability emerge. However, incorporating SFT data with the LoT-TL training method yields a marked improvement, demonstrating the effectiveness of transfer learning in equipping models with Lean4 Long CoT capabilities. Interestingly, additional training with correction data, although it is not designed for whole-proof writing, further enhances the performance. This improvement likely arises from the model developing self-analysis capabilities of Lean4 code, allowing it to avoid potentially wrong solutions.

### 3.5.2. SWITCH-OFF LONG CoT

This experiment shows that the strong FL reasoning power of *LoT-Solver* comes from the emergent formal reasoning ability in Long CoT, rather than trivially stacking more data. It uses our *LoT-Solver* model to write Lean4 proofs directly using the code completion method without Long CoT. We find the performance drop from 51.64% to 49.18%, which is because the model does not take explicit high-level plan in Long CoT, making it unable to finish some questions in the induction field.

### 3.5.3. ABLATION OF LONG CoT

To validate the quality of integration between NL and FL in Long CoT, we fine-tune the DeepSeek-Prover-v1.5-RL model directly using our SFT dataset without Long CoT reasoning. We can find that the performance of w/o Long CoT model (48.36%) is lower than *LoT-Solver* (51.64%). This confirms that Long CoT plays a crucial role in Lean4 theorem proving, offering structured reasoning that outperforms direct RL-based additional data fine-tuned model.

## 3.6. Case Study

This section presents the general results of case studies of **MA-LoT** framework. Due to the limited space, we leave detailed examples in Appendix D. The results show the collaboration between agents. The prover agent can use a high-level plan to prove advanced MATH theorems and the corrector can analyze the feedback from the Lean executor to formulate correct proof of IMO-level problem. The content in Long CoT also demonstrates the formal reasoning abilities that emerge from our *LoT-TL* training pipeline in both prover and corrector agents. These observations qualitatively validate the multi-agent system’s design and emergence of formal reasoning ability in Long CoT, demonstrating its ability to combine high-level planning with iterative refinement in Lean-based Long CoT.

## 4. Related Work

### 4.1. Lean4 Theorem Proving using LLMs

The application of LLMs for FL proving has been a hot topic for study these years. The tree-search including works

represented by Expert Iteration (Polu et al., 2022), Re-Prover (Yang et al., 2024b), Lean-Star (Lin et al., 2024), and InternLM-Step-Prover (Wu et al., 2024b). This direction does not take full consideration of the LLMs’ NL reasoning ability and costs exponentially increasing computation power. Another direction treats formal languages as code and asks the LLMs to do the complete-proof generation without interaction with the Lean executor to fully use the NL reasoning ability of LLMs. Significant works includes DeepSeek-Prover (Xin et al., 2024b;a), Theorem-Llama (Wang et al., 2024), and Llemma (Azerbayev et al., 2023). Works in this direction tend to overlook the verification signals from Lean executors or do not have thorough thinking of the error messages.

### 4.2. Agent-based LLM

Traditional RL methods offer a training method solution for general reasoning and decision-making processes but often suffer from low sample efficiency and generalization problems (Pourchot & Sigaud, 2018). With the fast-developing reasoning and instruction-following ability of LLMs, many researchers began to make LLMs as agents (Wang et al., 2022; Ma et al., 2024). The primary method for using LLMs as agents is to design special prompts and in-context examples to let LLMs interact with the outsourcing tools using actionable responses (Xie et al., 2023; Yang et al., 2023). Further efforts were made to apply specialized training to enhance their agentic capabilities (Xu et al., 2023; Reed et al., 2022). In the context of formal reasoning, most tree-search methods (Yang et al., 2024b; Lin et al., 2024; Wu et al., 2024a) apply an LLM as an agent query the executor and receive feedback to further refine the proof. Because of the huge difference between FL and NL, such methods are unable to provide high-level analysis to the problem and provide a structured response.

## 5. Conclusion

This paper introduces **MA-LoT**, (to the best of our knowledge), the first multi-agent Lean-based Long Chain-of-Thought framework for formal theorem proving. Our approach addresses the limitations of single-agent systems, which either under-utilize the NL reasoning and planning capabilities of LLMs or fail to integrate formal verification feedback effectively. By structuring interactions between a prover and a corrector agent through Long CoT, **MA-LoT** enables deeper insights and long-term coherence in proof generation. To support this framework, we propose *LoT-TL*, a training pipeline that makes formal reasoning Long CoT capability emerge to LLMs without requiring annotated Lean Long CoT data. Through extensive experiments on the MiniF2F-Test benchmark, **MA-LoT** achieves 54.51% accuracy, surpassing all baselines, including both tree-search



and whole-proof generation methods. These results underscore the advantages of integrating formal verification with structured reasoning leading to better Lean4 theorem proving capability. Beyond theorem proving, *LoT-TL* training pipeline demonstrates a potential method for applying Long CoT techniques to domain-specific tasks without specialized annotations. Additionally, the success of multi-agent Long CoT in Lean4 suggests broader applications of formal verification for enhancing structured reasoning across diverse fields.

## Impact statement

This paper presents work whose goal is to advance the formal theorem proving using LLMs. The potential social impact is majorly in the field of education. With the increasing number of formal languages used in graduate-level education, a more advanced formal theorem proving model may result in educators being unable to distinguish the model-generated results and student writing results. Despite the societal consequences of improving formal reasoning systems, specific discussions of ethical concerns are still too early at this stage.

## References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Azerbaiyev, Z., Schoelkopf, H., Paster, K., Santos, M. D., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.
- De Moura, L., Kong, S., Avigad, J., Van Doorn, F., and von Raumer, J. The lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pp. 378–388. Springer, 2015.
- Frieder, S., Bayer, J., Collins, K. M., Berner, J., Loader, J., Juhász, A., Ruehle, F., Welleck, S., Poesia, G., Griffiths, R.-R., et al. Data for mathematical copilots: Better ways of presenting proofs for machine learning. *arXiv preprint arXiv:2412.15184*, 2024.
- Harrison, J. Hol light: An overview. In *International Conference on Theorem Proving in Higher Order Logics*, pp. 60–66. Springer, 2009.
- 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. *arXiv preprint arXiv:2103.03874*, 2021.
- Jiang, A. Q., Li, W., Han, J. M., and Wu, Y. Lisa: Language models of isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pp. 378–392, 2021.
- 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. *arXiv preprint arXiv:2210.12283*, 2022.
- Kumarappan, A., Tiwari, M., Song, P., George, R. J., Xiao, C., and Anandkumar, A. Leanagent: Lifelong learning for formal theorem proving. *arXiv preprint arXiv:2410.06209*, 2024.
- Lin, H., Sun, Z., Yang, Y., and Welleck, S. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.
- Ma, C., Zhang, J., Zhu, Z., Yang, C., Yang, Y., Jin, Y., Lan, Z., Kong, L., and He, J. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024.

- 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* 28, pp. 625–635. Springer, 2021.
- Newell, A. and Simon, H. The logic theory machine—a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79, 1956.
- Open-Source-O1. Open-o1, 2024. URL <https://github.com/Open-Source-O1/Open-O1>. Accessed: 2024-12-28.
- OpenAI. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>, September 13 2024. Accessed: 2024-11-24.
- Paulson, L. C. *Isabelle: A generic theorem prover*. Springer, 1994.
- Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.
- Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.
- Pourchot, A. and Sigaud, O. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*, 2018.
- Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.
- Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D., Lillcrap, T., Alayrac, J.-b., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.
- Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Wang, R., Jansen, P., Côté, M.-A., and Ammanabrolu, P. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.
- Wang, R., Zhou, W., and Sachan, M. Let’s synthesize step by step: Iterative dataset synthesis with large language models by extrapolating errors from small models. *arXiv preprint arXiv:2310.13671*, 2023.
- Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*, 2024.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- Wu, Z., Huang, S., Zhou, Z., Ying, H., Wang, J., Lin, D., and Chen, K. Internlm2. 5-step-prover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 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.
- Xie, T., Zhou, F., Cheng, Z., Shi, P., Weng, L., Liu, Y., Hua, T. J., Zhao, J., Liu, Q., Liu, C., et al. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*, 2023.
- Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024a.
- Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., et al. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.
- Xu, Y., Su, H., Xing, C., Mi, B., Liu, Q., Shi, W., Hui, B., Zhou, F., Liu, Y., Xie, T., et al. Lemur: Harmonizing natural language and code for language agents. *arXiv preprint arXiv:2310.06830*, 2023.
- Yang, H., Yue, S., and He, Y. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.
- Yang, K., Poesia, G., He, J., Li, W., Lauter, K., Chaudhuri, S., and Song, D. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*, 2024a.
- 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. *Advances in Neural Information Processing Systems*, 36, 2024b.
- Ying, H., Wu, Z., Geng, Y., Wang, J., Lin, D., and Chen, K. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*, 2024.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

## A. Term chart

To make the reader better understand the terms, we provide this chart that explains every term, abbreviation, and corresponding tool in detail.

1. **NL** (Natural Language): Refers to language that humans use in our daily life, often unable to perform auto-verification.
2. **FL** (Formal Language): A structured and mathematically precise representation of logic and proofs, which ensures rigorous verification and eliminates ambiguities present in NL reasoning.
3. **Lean4**: A functional programming language and interactive theorem prover developed for formalizing mathematics and verifying proofs.
4. **Lean Executor**: The built-in proof verification engine of Lean4. It evaluates proof steps, checks for correctness, and ensures that every logical inference follows strict formal verification rules.
5. **Long CoT (Long Chain-of-Thought)**: The reasoning structure provided by OpenAI-O1 (OpenAI, 2024) that performs long and detailed thinking before making the final output. Different from standard CoT, Long CoT allows for multi-step logical reasoning before proof generation, reflection, and iterative refinement from self-check of Lean4 feedback.
6. **MA-LoT** (Multi-Agent Lean-based Long Chain-of-Thought framework): Our proposed multi-agent framework for formal theorem proving.
7. **LoT-TL** (LoT-Transfer Learning): The transfer learning pipeline we propose to enable LLMs with Lean4 Long CoT capability without the need for the specially annotated dataset.
8. **LoT-Solver**: The model we train through *LoT-TL* pipeline that serves both as prover and corrector agent. It is a high-standard Lean4 theorem proving model with Long CoT capability to control agent type and enhance formal thinking.
9. **LoT-PD & LoT-CD** (LoT-ProveData) & (LoT-CorrectionData): The LoT-PD is a dataset containing verified Lean4 theorem proofs and NL statement and proof. Its main function is to provide basic capability for the prover agent. The LoT-CD is a dataset containing incorrect error messages and correct Lean4 proof together with NL annotations. It majorly provides capability for the prover agent to refine proofs.

Training Steps	MiniF2F-Test
<b>125 Steps</b>	32.79%
<b>250 Steps</b>	35.25%
<b>1,250 Steps</b>	45.08%
<b>2,500 Steps</b>	47.54%
<b>13,616 Steps (full training)</b>	51.64%

Table 4. MiniF2F-Test result on different data-scale trained models

## B. Examples for tree-search and whole-proof generation

Following Section 2.1, we present the input-output example for tree-search method and whole-proof generation method here.

Example for whole-proof generation:

```
### Input Example
Complete the following Lean 4 code:
theorem algebra_sqineq_unitcircatbpbmlt1
  (a b: ℝ)
  (h₀ : a^2 + b^2 = 1) :
  a * b + (a - b) ≤ 1 := by
### Output Example
-- We have that (a - b - 1)^2 ≥ 0.
have h₁ : 0 ≤ (a - b - 1) ^ 2 :=
  sq_nonneg _
-- By expanding, we have:
-- 0 ≤ a^2 - ab - a - ab + b^2 + b - a + b + 1.
linarith [h₀, sub_add_cancel a b]
```

Example for tree-search method:

```
### Input Example
DECL MyNat.mul_pow
GOAL a b n : N
⊢ (a * b) ^ n = a ^ n * b ^ n
### Output Example
PROOFSTEP induction n with t Ht
```

## C. Scaling Law Study

We conduct an additional experiment to investigate whether our model adheres to scaling laws by evaluating its performance across varying training dataset scales. Specifically, we train the model with four distinct step sizes (125, 250, 1250, and 2500 steps) under a fixed batch size of 4 and evaluate its accuracy on the MiniF2F-Test benchmark using the Long Chain-of-Thought (CoT) format. The resulting accuracy rates are presented in Table 4. To analyze these results, we plot the relationship between log-scaled training steps and model accuracy in Figure 3. A linear approximation of the data yields a strong coefficient of determination ( $R^2 = 0.9664$ ), indicating a near-linear improvement in performance as training steps increase logarithmically. This aligns with the predictions of scaling laws, suggesting that our approach benefits significantly from larger-scale train-



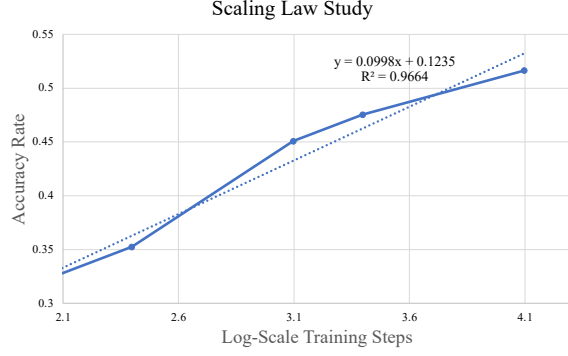


Figure 3. Log-scale training steps and accuracy rate on MiniF2F of trained model. The dashed line is linear regression approximation line

ing. Our findings further highlight the potential for enhanced performance with expanded resources to annotate additional Lean4 data.

## D. Case studies

We present two case studies in Appendix D.1 and D.2. We can see that the multi-agent framework is able to write a more coherent and well-structured proof by making a high-level NL plan and executing such plan in the prover agent. The corrector agent is able to analyze the unsolved goals, make plans for correction, and write correct proof. This is achieved by our proposed training procedure.

### D.1. Prover analysis

This is the example of provers agent generated result, in which we can see that the agent learns to write a high-level proof plan in the Long CoT and details execute the plan when writing the Lean4 code

```

### Provided Long CoT Header
<Thought>
Alright, I should do the following:
  1. Provide the natural language analysis for the theorem based on the Natural
    ↪ language theorem statement.
  2. Draft the Lean4 tactics I should use to solve the problem
  3. Write the output Lean4 code.
The user also asks that I should avoid using the keyword `sorry` to give up the
↪ proof, so I will not write it in my Lean4 code.
### Generated results:
The `mathd_numbertheory_458` can be proofed by leveraging the properties of
↪ modular arithmetic. Specifically, we know that if  $n \% 8 = 7$ , then  $n$  can
↪ be expressed as  $n = 8k + 7$  for some integer  $k$ . We need to find  $n \% 4$ .
First, observe that  $n \% 8 = 7$  implies  $n = 8k + 7$ . Since  $8k$  is divisible by
↪  $4$ , we only need to consider the remainder when  $7$  is divided by  $4$ , which
↪ is  $3$ . Therefore,  $n \% 4 = 3$ .
Now, translating this reasoning into Lean4 code, we can use the properties of
↪ modular arithmetic to directly derive the result.
</Thought>
<Output>
```lean4
theorem mathd_numbertheory_458 (n : ℕ) (h₀ : n % 8 = 7) : n % 4 = 3 := by
  -- We know that n leaves a remainder of 7 when divided by 8.
  have h₁ : n % 8 = 7 := h₀
  -- We need to find the remainder when n is divided by 4.
  have h₂ : n % 4 = 3 := by
    -- Using the fact that 4 divides 8, we can use the modulo operation
    ↪ properties.
    rw [← Nat.mod_mod_of_dvd n (by decide : 4 ∣ 8)]
    -- Simplify the expression using the given remainder when divided by 8.
    simp [h₁]
    -- Therefore, the remainder when n is divided by 4 is 3.
  exact h₂
```
</Output>

```

### D.2. Corrector analysis

This is an example of error correction that successfully solve an IMO problem, we can see that the model firstly analysis the error message, then make proof plan, and then write correct proof to another IMO-level problem

```

### Provided Long CoT Header
<Thought>
Alright, I need to prove the theorem prover_data_68493 using the Lean4 code. Here
↪ is my draft of the proof:
```lean4
theorem imo_1983_p6
  (a b c : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)

```

```

(h1 : c < a + b)
(h2 : b < a + c)
(h3 : a < b + c) :
0 ≤ a2 * b * (a - b) + b2 * c * (b - c) + c2 * a * (c - a) := by
/-
To prove that  $\forall (a^2 b(a - b) + b^2 c(b - c) + c^2 a(c - a) \geq 0)$  for
→ positive real numbers  $\forall (a)$ ,  $\forall (b)$ , and  $\forall (c)$  satisfying certain
→ conditions related to being part of a triangle, we can proceed as follows:
1. Nonlinear Arithmetic: Utilize nonlinear arithmetic (nlinarith) to
→ verify inequalities involving squares and products of differences between
→ variables. These operations inherently handle cases where expressions might
→ not hold due to cancellation or sign changes.
Equality holds if all pairwise differences among  $\forall (a)$ ,  $\forall (b)$ , and  $\forall (c)$ 
→ become zero simultaneously. However, since they must satisfy strict
→ positivity constraints imposed by triangle, such an eventuality cannot
→ occur under normal circumstances unless specified otherwise. Nonetheless,
→ within the confines of algebraic manipulation without additional
→ constraints beyond those provided, no immediate contradictions arise
→ regarding potential zeros leading to equality.
-/
-- Introduce intermediate hypotheses ensuring divisibility properties do not
→ disrupt expected behavior.
have ha' : True := trivial
have hb' : True := trivial
have hc' : True := trivial
-- Employ nonlinear arithmetic to iteratively check for inequalities involving
→ sums and products of our main variables and constants.
repeat' nlinarith [mul_pos h0.1 h0.2.1, mul_pos h0.2.1 h0.2.2, mul_pos h0.2.2
→ h0.1,
  sub_self a, sub_self b, sub_self c, add_sub_cancel'_right a b,
  → add_sub_cancel'_right b c,
  add_sub_cancel'_right c a, mul_nonneg (ha'.le) (hb'.le), mul_nonneg (hb'.le)
  → (hc'.le),
  mul_nonneg (hc'.le) (ha'.le)]
```&
Let me test it in Lean4
Emmm, it seems the above proof is wrong.
Let me check the error messages.
OK, Here is the error messages:
```bash
line 13

unsolved goals
a b c : ℝ
h0 : 0 < a ∧ 0 < b ∧ 0 < c
h1 : c < a + b
h2 : b < a + c
h3 : a < b + c
ha' hb' hc' : True
⊢ 0 ≤ a2 * b * (a - b) + b2 * c * (b - c) + c2 * a * (c - a)
```&
So, I will rethink a Lean4 proof following the steps

```

1. Provide the natural language analysis for the theorem based on the Natural
  - language theorem statement, Lean4 theorem statement, my previous proof and
  - the error message.
2. Draft the Lean4 tactics I should use to solve the problem
3. Write the output Lean4 code.

Let me analysis the wrong Lean4 solution through the error messages.

### Generated results"

The error message indicates that the proof is incomplete.

The error message says:

```

unsolved goals

a b c : ℝ

h<sub>0</sub> : 0 < a ∧ 0 < b ∧ 0 < c

h<sub>1</sub> : c < a + b

h<sub>2</sub> : b < a + c

h<sub>3</sub> : a < b + c

ha' hb' hc' : True

⊢ 0 ≤ a<sup>2</sup> \* b \* (a - b) + b<sup>2</sup> \* c \* (b - c) + c<sup>2</sup> \* a \* (c - a)

```&

So, the main issue is that the proof is incomplete, and the final goal  $0 \leq a^2 * b * (a - b) + b^2 * c * (b - c) + c^2 * a * (c - a)$  is not being addressed properly.

Therefore, I need to construct a proper proof strategy that addresses all the conditions and goals properly.

First, I need to understand the mathematical structure of the inequality  $a^2 * b * (a - b) + b^2 * c * (b - c) + c^2 * a * (c - a)$ .

This expression is symmetric in  $a$ ,  $b$ , and  $c$ .

Given that  $a$ ,  $b$ , and  $c$  are positive real numbers and they satisfy certain conditions related to being part of a triangle (triangle inequalities), I can think about the properties of such expressions.

One approach is to consider the symmetry and positivity of  $a$ ,  $b$ , and  $c$ .

Since  $a$ ,  $b$ , and  $c$  are sides of a triangle, they satisfy the triangle inequalities:

- $a + b > c$
- $b + c > a$
- $c + a > b$

Additionally, since  $a$ ,  $b$ , and  $c$  are positive, we can use properties of real numbers and inequalities to prove the desired result.

Another approach is to use algebraic manipulation or inequalities like AM-GM inequality or Cauchy-Schwarz inequality.

Given the complexity of the expression, perhaps rearranging terms or applying known inequalities could help.

Also, considering the conditions of the triangle, perhaps expressing the variables in terms of each other or using substitution could simplify the problem.

I need to ensure that the final proof is logically sound and does not rely on assumptions that are not justified.

Therefore, I will proceed to construct a revised proof based on these considerations.

Let me proceed with the thought process and output the corrected Lean4 code.

</Thought>

<Output>

```lean4



```

/-- Let $a$, $b$, and $c$ be the lengths of the sides of a triangle. Prove that
$a^2 b(a - b) + b^2 c(b - c) + c^2 a(c - a) \geq 0$.
-/
theorem imo_1983_p6
  (a b c : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
  (h₁ : c < a + b)
  (h₂ : b < a + c)
  (h₃ : a < b + c) :
  0 ≤ a^2 * b * (a - b) + b^2 * c * (b - c) + c^2 * a * (c - a) := by
  -- We will use the fact that the square of any real number is non-negative.
  have h₄ : 0 ≤ (a - b)^2 := sq_nonneg (a - b)
  have h₅ : 0 ≤ (b - c)^2 := sq_nonneg (b - c)
  have h₆ : 0 ≤ (c - a)^2 := sq_nonneg (c - a)
  -- Using these inequalities, we can conclude the desired inequality using
  → non-linear arithmetic.
  nlinarith [mul_pos h₀.1 h₀.2.1, mul_pos h₀.2.1 h₀.2.2, mul_pos h₀.2.2 h₀.1,
    mul_pos (sub_pos.mpr h₁) (sub_pos.mpr h₂), mul_pos (sub_pos.mpr h₂)
      → (sub_pos.mpr h₃),
    mul_pos (sub_pos.mpr h₃) (sub_pos.mpr h₁)]
  <`&
</Output>

```

## E. Training examples

This section we present the training examples for SFT data and correction data. We can see that in the LoT-TL training stage, we keep the system prompt to switch off the Long CoT thinking. In particular, in the SFT data, we switch off the entire Long CoT, in the correction data training, we only switch off the analysis of error message in the Long CoT while provide the error message as well as the original proof draft as a component in the Long CoT.

### E.1. LoT-TL Training example for SFT data

```

=== Input Example ===
<|begin_of_sentence|>You are a helpful assistant who will solve every problem
→ **WITHOUT** Long Chain-of-Thought
### Instruction:
@ Natural language theorem statement:
prover_data_298:
This all simplifies to: $\frac{55*35}{12^7}$
@ Lean4 theorem statement:
```lean4
theorem prover_data_298 :
  ((55 * 35) / (12^7) : ℚ) = (55 * 35) / (12^7) := by
  <`&
@ Lean4 theorem statement and proof with explanatory comments preceding each
→ line of code:
### Response:
=== Outpus Example ===
<Thought>
The user ask not to solve with long CoT, so I will directly write the answer.
</Thought>
<Output>
```lean4
/--This all simplifies to: $\frac{55*35}{12^7}$-/

```

```

theorem prover_data_298 :
  ((55 * 35) / (12^7) : ℚ) = (55 * 35) / (12^7) := by
  /-
  To prove that  $\frac{55 \cdot 35}{12^7}$  is equal to  $\frac{55 \cdot 35}{12^7}$ , we simply observe that the expressions on both sides of
  → the equality are identical. This identity holds true because the
  → numerator and denominator are the same on both sides.
  -/
  -- We use the fact that the left-hand side and the right-hand side of the
  → equality are the same.
  congr 1
  -- Simplify the power expression.
  <;> simp [pow_succ]
  -- Normalize the numerical values to confirm the equality.
  <;> norm_num
  ```&
</Output><|end_of_sentence|>

```

## E.2. LoT-TL Training example for correcton data

```

=== Input Example ===
<|begin_of_sentence|>You are a helpful assistant who will solve every problem
→ **WITH** Long Chain-of-Thought. You can make some Lean4 proof draft in
→ your O1 style Long internal Chain of Thought (CoT) and you have the
→ ability to test the thing during your thinking. You need to use the error
→ messages in your Long CoT to guide your proof generation but you are
→ required **NOT** to provide natural language analysis to the errors.
### Instruction:
@ Natural language theorem statement:
prover_data_2216:
Given non-negative real numbers $a, b, c$ with $a + b + c = 1$. Prove that:
→  $7(ab + bc + ca) \leq 2 + 9abc$ .
@ Lean4 theorem statement:
```lean4
theorem prover_data_2216 (a b c : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) (hc : 0 ≤ c)
→ (habc : a + b + c = 1) : 7 * (a * b + b * c + c * a) ≤ 2 + 9 * a * b * c
→ := by
```&
@ Lean4 theorem statement and proof with explanatory comments preceding each
→ line of code:
### Response:
<Thought>
Alright, I need to prove the theorem prover_data_2216 using the Lean4 code.
→ Here is my draft of the proof:
```lean4
theorem prover_data_2216 (a b c : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) (hc : 0 ≤ c)
→ (habc : a + b + c = 1) : 7 * (a * b + b * c + c * a) ≤ 2 + 9 * a * b * c
→ := by
  with aes
```&
Let me test it in Lean4
Emmm, it seems the above proof is wrong.
Let me check the error messages.
OK, Here is the error messages:

```

```

```bash
line 7
unexpected token 'with'; expected '{' or tactic
```&
```bash
line 7
unsolved goals
a b c : ℝ
ha : 0 ≤ a
hb : 0 ≤ b
hc : 0 ≤ c
habc : a + b + c = 1
⊢ 7 * (a * b + b * c + c * a) ≤ 2 + 9 * a * b * c
```&
=== Output Example ===
Since the user ask NOT to provide natural language analysis to the problem.
→ Let me directly try to write the revised code.
<\Thought>
<Output>
```lean4
/--Given non-negative real numbers $a, b, c$ with $a + b + c = 1$. Prove
→ that: $7(ab + bc + ca) \leq 2 + 9abc$.-/
theorem prover_data_2216 (a b c : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) (hc : 0 ≤ c)
→ (habc : a + b + c = 1) : 7 * (a * b + b * c + c * a) ≤ 2 + 9 * a * b * c
→ := by
/-
Given non-negative real numbers \a\, \b\, and \c\ such that \a + b +
→ c = 1\, we need to prove that \7(ab + bc + ca) \leq 2 + 9abc\). The
→ proof involves using algebraic manipulations and inequalities to show
→ that the left-hand side is bounded above by the right-hand side.
→ Specifically, we use the fact that squares of real numbers are
→ non-negative, which helps in establishing the required inequality.
-/
-- Normalize the expression involving `a`, `b`, and `c` to prepare for
→ algebraic manipulation.
ring_nf at habc ⊢
-- Use `nlinarith` to handle nonlinear arithmetic inequalities. This
→ includes checking non-negativity of expressions involving `a`, `b`, and
→ `c`.
nlinarith [sq_nonneg (a - 1), sq_nonneg (b - 1), sq_nonneg (c - 1),
sq_nonneg (a - b), sq_nonneg (b - c), sq_nonneg (c - a),
sq_nonneg (a + b + c)]
```&
</Output><|end_of_sentence|>

```

## F. Agents input example

### F.1. Prover agent input-output example

```
### Input Example
IMO problem our prover found
@ Natural language theorem statement:
imo_1961_p1:
(''Hungary'')
Solve the system of equations:
```

```
<center>
$
\begin{matrix}
\quad x + y + z \quad \&= \quad a \quad ; \quad \backslash, \quad \backslash \\
x^2 + y^2 + z^2 \quad \&= \quad b^2 \quad \backslash \\
\quad \quad \quad xy \quad \backslash \quad \quad \quad \&= \quad z^2
\end{matrix}
$
</center>
```

where \$a\$ and \$b\$ are constants. Give the conditions that \$a\$ and \$b\$  
 $\rightarrow$  must satisfy so that \$x, y, z\$ (the solutions of the system) are  
 $\rightarrow$  distinct positive numbers.

@ Lean4 theorem statement:

```
```lean4
theorem imo_1961_p1 (x y z a b : ℝ) (h₀ : 0 < x ∧ 0 < y ∧ 0 < z) (h₁ : x ≠ y)
 $\rightarrow$  (h₂ : y ≠ z)
(h₃ : z ≠ x) (h₄ : x + y + z = a) (h₅ : x ^ 2 + y ^ 2 + z ^ 2 = b ^ 2) (h₆ :
 $\rightarrow$  x * y = z ^ 2) :
0 < a ∧ b ^ 2 < a ^ 2 ∧ a ^ 2 < 3 * b ^ 2 := by
```&
```

@ Lean4 theorem statement and proof with explanatory comments preceding each  
 $\rightarrow$  line of code:

Response:

<Thought>

Alright, I should do the following:

1. Provide the natural language analysis for the theorem based on the  
 $\rightarrow$  Natural language theorem statement.
2. Draft the Lean4 tactics I should use to solve the problem
3. Write the output Lean4 code.

The user also asks that I should avoid using the keyword `sorry` to give up  
 $\rightarrow$  the proof, so I will not write it in my Lean4 code.

### Output Example

The `imo\_1961\_p1` can be proofed by analyzing the given equations and  
 $\rightarrow$  deriving necessary inequalities to ensure that the solutions are distinct  
 $\rightarrow$  positive numbers.

</Thought>

<Output>

```
```lean4
theorem imo_1961_p1 (x y z a b : ℝ) (h₀ : 0 < x ∧ 0 < y ∧ 0 < z) (h₁ : x ≠ y)
 $\rightarrow$  (h₂ : y ≠ z)
```



```

(h3 : z ≠ x) (h4 : x + y + z = a) (h5 : x ^ 2 + y ^ 2 + z ^ 2 = b ^ 2) (h6 :
  → x * y = z ^ 2) :
0 < a ∧ b ^ 2 < a ^ 2 ∧ a ^ 2 < 3 * b ^ 2 := by
-- We need to prove three inequalities: 0 < a, b^2 < a^2, and a^2 < 3 *
  → b^2.
refine' ⟨_, _, _⟩
-- First, prove 0 < a.
-- Since x, y, z are positive, their sum a must also be positive.
. nlinarith [h0.1, h0.2.1, h0.2.2, sq_nonneg (x - y), sq_nonneg (y - z),
  → sq_nonneg (z - x)]
-- Next, prove b^2 < a^2.
-- Using the given equations and inequalities, derive the necessary
  → inequalities.
. nlinarith [mul_self_pos.mpr (sub_ne_zero.mpr h1), mul_self_pos.mpr
  → (sub_ne_zero.mpr h2),
    mul_self_pos.mpr (sub_ne_zero.mpr h3), sq_nonneg (x - y), sq_nonneg (y -
  → z), sq_nonneg (z - x)]
-- Finally, prove a^2 < 3 * b^2.
-- Combine all the given information to derive this inequality.
. nlinarith [mul_self_pos.mpr (sub_ne_zero.mpr h1), mul_self_pos.mpr
  → (sub_ne_zero.mpr h2),
    mul_self_pos.mpr (sub_ne_zero.mpr h3), sq_nonneg (x - y), sq_nonneg (y -
  → z), sq_nonneg (z - x)]
```&
</Output>

```

## F.2. Corrector agent input-output example

```

### Input Example
You are a helpful mathematical assistant specialized in formal theorem
  → proving using Lean4.
Your objectives:
1. Read and interpret the Lean4 theorem statement and any error messages.
2. **If a previous proof attempt was incorrect, analyze its exact mistakes
  → and completely discard or rewrite the proof as needed.**
3. **Avoid reusing incorrect proof structures or strategies unless explicitly
  → validated as correct.**
4. **Address all error messages** by modifying the proof structure as needed.
5. Provide a detailed thought process in the <Thought> section, but **only
  → place the corrected Lean4 code in the <Output> section**.
6. **Ensure the new proof is logically valid and does not use `sorry`.**
  → Instruction:Below are some **correct Lean4 theorem proof examples** for
  → your reference. Use them as guidance when constructing the revised proof.
  → Ensure that your final proof aligns with these examples in terms of
  → clarity, structure, and correctness. Instruction: @ Natural language
  → theorem statement:
mathd_algebra_392
The sum of the squares of three consecutive positive even numbers is $12296$.
  → Find the product of the three numbers divided by $8$. Show that it is
  → 32736.

@ Lean4 theorem statement:
```lean4
theorem mathd_algebra_392

```

```

(n : ℕ)
(h₀ : Even n)
(h₁ : ((n:ℤ) - 2)^2 + (n:ℤ)^2 + ((n:ℤ) + 2)^2 = 12296) :
((n - 2) * n * (n + 2)) / 8 = 32736 :=
```&

@ Lean4 theorem statement and proof with explanatory comments preceding each
→ line of code:
Response:
<Thought>
Alright, I need to prove the theorem prover_data_68493 using the Lean4 code.
→ Here is my draft of the proof:

```lean4
theorem mathd_algebra_392
  (n : ℕ)
  (h₀ : Even n)
  (h₁ : ((n:ℤ) - 2)^2 + (n:ℤ)^2 + ((n:ℤ) + 2)^2 = 12296) :
  ((n - 2) * n * (n + 2)) / 8 = 32736 := by
  -- Convert the equation to integers for easier manipulation
  norm_cast at h₁
  -- Expand and simplify the given equation
  ring_nf at h₁
  -- Solve for n using numerical reasoning
  have h₂ : n = 24 := by
    nlinarith
  -- Substitute n = 24 into the expression and simplify
  norm_num [h₂]
```&

Let me test it in Lean4
Emmm, it seems the above proof is wrong.
Let me check the error messages.
OK, Here is the error messages:
```bash
line 18

linarith failed to find a contradiction
case h1.h
n : ℕ
h₀ : Even n
h₁ : Int.subNatNat n 2 ^ 2 + ↑(n ^ 2) + ↑(4 + n * 4 + n ^ 2) = 12296
a↑ : n < 24
⊢ False
failed
```&
```bash
line 11

unsolved goals
n : ℕ
h₀ : Even n
h₁ : Int.subNatNat n 2 ^ 2 + ↑(n ^ 2) + ↑(4 + n * 4 + n ^ 2) = 12296
h₂ : n = 24
⊢ False

```

```
```&
```

So, I will rethink a Lean4 proof following the steps

1. Provide the natural language analysis for the theorem based on the
  - Natural language theorem statement, Lean4 theorem statement, my
  - previous proof and the error message.
2. Draft the Lean4 tactics I should use to solve the problem
3. Write the output Lean4 code.

Let me analysis the wrong Lean4 solution through the error messages.

### Output Example

From the error messages, it seems the following points:

- `linarith failed to find a contradiction` suggests that the linear
  - arithmetic tactic was unable to derive a contradiction from the given
  - hypotheses, indicating that the assumptions might not lead to a
  - contradiction as expected.
- `case h1.h` indicates a specific case within a larger context where a
  - contradiction was expected but was not found, suggesting that the
  - assumption `n < 24` might not be valid under certain conditions.
- `unsolved goals` means that after applying the tactics, there are still
  - unresolved goals left, indicating that the tactics applied were
  - insufficient to complete the proof.

From these error messages, I realize that my previous approach might not have
 

- handled the specifics of the problem correctly. Therefore, I need to
- revise my proof strategy.

First, I will analyze the natural language theorem statement and the provided
 

- Lean4 theorem statement to understand the problem fully.

The problem involves three consecutive even numbers whose squares sum up to
 

- 12296. The goal is to find the product of these three numbers divided by
- 8 and show that it equals 32736.

Given that the numbers are consecutive even numbers, I can denote the middle
 

- number as `n`. Then, the three consecutive even numbers can be
- represented as `n-2`, `n`, and `n+2`.

The sum of their squares is given by:

```
...
((n - 2)^2) + (n^2) + ((n + 2)^2) = 12296
...
```

Expanding these squares, we get:

```
...
(n^2 - 4n + 4) + n^2 + (n^2 + 4n + 4) = 12296
...
```

Combining like terms:

```
...
3n^2 + 8 = 12296
...
```

Subtracting 8 from both sides:

```
...
3n^2 = 12288
...
```

Dividing both sides by 3:

```
...
n^2 = 4096
...
```

Taking the square root of both sides:

```
...
```

```

n = 64
...
Since we are dealing with squares, we take the positive root because we are
↪ dealing with natural numbers.
Now, the three consecutive even numbers are:
...
n - 2 = 62
n = 64
n + 2 = 66
...
The product of these numbers is:
...
62 * 64 * 66
...
To compute this product, I can break it down into smaller factors:
...
62 * 64 = 3968
3968 * 66 = 3968 * (60 + 6) = 3968 * 60 + 3968 * 6 = 238080 + 23808 = 261888
...
Now, divide this product by 8:
...
261888 / 8 = 32736
...
Thus, the final result matches the expected outcome.
Now, I need to translate this reasoning into Lean4 code.
I will follow these steps:
1. Define the theorem with the given parameters.
2. Convert the given integer expressions to natural numbers.
3. Expand and simplify the equation using algebraic manipulation.
4. Solve for `n` using numerical methods.
5. Compute the product of the three numbers and verify that dividing by 8
↪ gives the desired result.
I will also include necessary imports and use appropriate tactics to handle
↪ the calculations.
Now, I will proceed to write the Lean4 code following this plan.
</Thought>

```

<Output>

```

```lean4
theorem mathd_algebra_392
  (n : ℕ)
  (h₀ : Even n)
  (h₁ : ((n : ℤ) - 2)^2 + (n : ℤ)^2 + ((n : ℤ) + 2)^2 = 12296) :
  ((n - 2) * n * (n + 2)) / 8 = 32736 := by
  -- Convert the equation to integers for easier manipulation
  push_cast at h₁
  -- Expand and simplify the given equation
  ring_nf at h₁
  -- Solve for n using numerical reasoning
  have h₂ : n = 64 := by
    nlinarith
  -- Substitute n = 64 into the expression and simplify
  subst h₂
  norm_num

```

```
```&  
</Output>
```