Hierarchical Attention Generates Better Proofs

Jianlong Chen[1,2], Chao Li[2], Yang Yuan[2,3†], Andrew C Yao[2,3†]

[1]The Chinese University of Hong Kong, Shenzhen
[2]Shanghai Qi Zhi Institute [3]IIIS, Tsinghua University
{jianlongchen}@link.cuhk.edu.cn, {yuanyang,andrewcyao}@tsinghua.edu.cn

Abstract

Large language models (LLMs) have shown promise in formal theorem proving, but their token-level processing often fails to capture the inherent hierarchical nature of mathematical proofs. We introduce **Hierarchical Attention**, a regularization method that aligns LLMs' attention mechanisms with mathematical reasoning structures. Our approach establishes a five-level hierarchy from foundational elements to high-level concepts, ensuring structured information flow in proof generation. Experiments demonstrate that our method improves proof success rates by 2.05% on miniF2F and 1.69% on ProofNet while reducing proof complexity by 23.81% and 16.50% respectively. The code is available at `https://github.com/Car-pe/HAGBP`.

1 Introduction

The intersection of AI and mathematics has emerged as an important research direction in recent years, particularly in the domain of formal theorem proving. Proof assistants, such as Lean (De Moura et al., 2015; Moura and Ullrich, 2021), Coq (The Coq Development Team, 2024), and Isabelle (Paulson, 1994), have become key platforms to explore this direction. Traditionally, theorem provers primarily rely on search-based methods to systematically explore proof spaces, often guided by complex rule-based techniques or symbolic heuristics (Han et al., 2021; Jiang et al., 2021; Polu and Sutskever, 2020; Polu et al., 2022; Lample et al., 2022; Jiang et al., 2022b; Yang et al., 2024).

The advent of large language models (LLMs) has brought a transformative shift, leveraging their capacity for deep contextual understanding to reason about mathematical proofs (Xin et al., 2024; Welleck and Saha, 2023; Zhao et al., 2023; Jiang et al., 2023; Wang et al., 2023a; First et al., 2023). These models excel at generating proofs and tackling a broad array of problems, significantly reducing the need for manually crafted heuristics. However, they still struggle with key challenges in formal theorem proving, often failing to generate difficult proofs or producing unnecessarily long ones.

These limitations arise because mathematics is inherently formal and rigorous, whereas LLMs are primarily designed to process plain token sequences, without explicit formal semantics. Therefore, the structured nature of formal concepts — where dependencies and relationships between concepts play a critical role — is difficult for LLMs to fully capture. This raises a natural question:

How to understand structure better?

Mathematical theorem proving exhibits inherent hierarchical structures in the flow of information between different components. While large language models have shown promising results in this domain, their attention mechanisms often fail to capture these natural hierarchies. We propose a novel framework that guides the model's attention patterns to better align with the hierarchical nature of mathematical reasoning, while maintaining flexibility for complex proof steps.

Our key insight is that mathematical reasoning follows a natural hierarchical structure, with information flowing from foundational elements to higher-level concepts. As shown in Figure 1, we formalize this intuition through a five-level hierarchy and implement it by structured attention patterns. This hierarchical framework not only respects the natural dependencies in mathematical proofs but also provides flexibility in attention distribution, allowing the model to capture both local and cross-level relationships necessary for complex reasoning.

Based on this framework, we propose **Hierar-**

[†]Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 17506–17520
July 27 - August 1, 2025

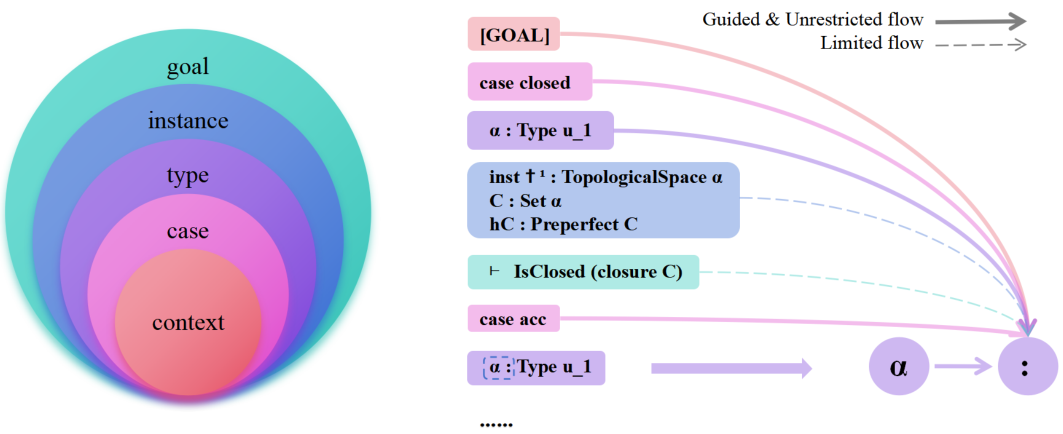

Figure 1: Overview of our hierarchical attention framework. **Left:** The five-level hierarchy from inner (context) to outer (goal) layer, illustrating the natural information flow in mathematical reasoning. **Right:** A concrete example showing how different components in a theorem proving state are assigned to hierarchical levels, with guided and unrestricted flow (solid arrows) representing allowed attention paths and limited flow (dashed arrows) representing restricted attention paths.

chical Attention, a novel regularization method aimed at improving structural learning in LLMs. Our approach constructs a hierarchical tree from the input token sequence, assigning levels to tokens and guiding information flow based on these levels. Specifically, we enforce the following constraints:

- Tokens at higher levels can access information from the same level or lower levels.
- Tokens at lower levels are restricted from accessing higher-level information.

Through extensive experiments on multiple theorem-proving benchmarks—including miniF2F (Zheng et al., 2021) and ProofNet (Azerbayev et al., 2023)—our method demonstrates significant improvements in both **proof success rates** and **proof conciseness**. Specifically, our approach achieves a 2.05% improvement in proof success rates while reducing the proof length by 23.81% in successful cases. These results highlight the advantages of preserving semantic and hierarchical structures in theorem proving. This is further confirmed by our ablation studies and attention pattern analysis.

The main contributions of this work are as follows:

- We analyzed the hierarchical structure in mathematical reasoning, from foundational definitions to final goals.
- We proposed a new algorithm for better structure learning for LLMs.
- We demonstrated substantial improvements on multiple standard benchmarks in proof accuracy and proof conciseness.

2 Related Work

Formal Theorem Proving. Formal theorem proving systems are typically classified into two categories: Automated Theorem Proving (ATP) and Interactive Theorem Proving (ITP). ATP systems aim to discover proofs without human intervention automatically. Saturation-based provers like E (Schulz, 2002) and Vampire (Kovács and Voronkov, 2013) use resolution calculus, while specialized solvers like SAT and SMT solvers (e.g., MiniSat (Eén and Sörensson, 2003), Z3 (De Moura and Bjørner, 2008)) focus on boolean satisfiability and other mathematical theories. Domain-specific systems like GEX (Chou et al., 2000) handle geometric problems through specialized deduction rules.

In contrast, ITP systems like Lean (De Moura et al., 2015; Moura and Ullrich, 2021), Coq (The Coq Development Team, 2024), and Isabelle (Paulson, 1994) emphasize human-machine collaboration. These systems provide expressive proof languages and sound kernels, enabling mathematicians to formalize theorems and construct proofs in a manner that mirrors informal mathematical reasoning while ensuring logical correctness.

Neural Theorem Proving. Neural Theorem Proving has risen to prominence alongside the rapid development of LLMs and more specialized neural architectures for formal reasoning. A central focus has been autoformalization (Wang et al., 2018, 2020; Wu et al., 2022b; Murphy et al., 2024; Jiang et al., 2022a, 2023; Lu et al., 2024; Ying et al., 2024a; Azerbayev et al., 2023; Liu et al., 2023; ?), which converts informal mathematical statements and proofs into machine-verifiable languages despite the ongoing challenges in semantic alignment. Another key area is premise selection (Irving et al., 2016; Kucik and Korovin, 2018; Piotrowski and Urban, 2020; Ferreira and Freitas, 2020a,b; Wu, 2022; Mikuła et al., 2023; Holden and Korovin, 2025), where models retrieve the most relevant lemmas from vast libraries to aid in proving a target statement. Researchers also tackle proof-step generation (Huang et al., 2018; Yang et al., 2024; Welleck and Saha, 2023; Sanchez-Stern et al., 2020, 2023; Yang and Deng, 2019; Polu and Sutskever, 2020; Han et al., 2021; Wang et al., 2023b, 2024; Lin et al., 2024; Wu et al., 2024; Rute et al., 2024; Dong and Ma, 2025; Lin et al., 2025; Wang et al., 2025b,a; Zhang et al., 2025), aiming to accurately predict the next formal step or tactic, often through auto-regressive models that learn from existing proofs. A further challenge is proof search (Loos et al., 2017; Suda, 2021; Aygün et al., 2020, 2022; Chvalovskỳ et al., 2023; Rawson and Reger, 2019, 2021; McKeown and Sutcliffe, 2023; Fokoue et al., 2023; Abdelaziz et al., 2022; Crouse et al., 2021; Xin et al., 2025), where deep learning-guided algorithms, sometimes using Monte Carlo Tree Search or reinforcement learning, explore and prune massive proof spaces, balancing correctness with computational efficiency.

Hierarchical Attention Mechanisms for Mathematical Reasoning. Mathematical documents typically have an implicit multilevel structure, from foundational definitions to the main theorems. Previous studies have attempted to exploit this hierarchical nature by parsing formulas or proofs into trees or graphs to better represent logical structures (Wang et al., 2017; Peng and Ma, 2017; Paliwal et al., 2020; Rawson and Reger, 2020), or by building dependency graphs over entire libraries to capture relationships between statements and lemmas (Ferreira and Freitas, 2020b; Bauer et al., 2024). These approaches, while promising, often depend on carefully crafted rules or programmatically generated data, lacking mechanisms to ensure that neural models respect the partial orders and compositional dependencies inherent in mathematical logic.

The attention mechanism is central to modern Transformer-based models (Vaswani, 2017). Although studies have explored their use in tasks such as generating math problems or document classification (Yang et al., 2016; Wu et al., 2022a), there is a gap in leveraging attention-based methods explicitly for mathematical reasoning.

3 Preliminaries

3.1 Hierarchical Structure in Lean

Lean is a strongly typed language, which allows all tokens to be naturally unfolded across multiple semantic levels. These levels align with various components of reasoning, with each successive level built upon the foundations of the preceding ones. The categorization of these layers can be delineated as follows:

Lowest or contextual layer: Contains background information, auxiliary concepts, or general knowledge relevant to the proof (T_0: context).

Intermediate layers: Include pattern matching and case analysis (T_1: case), type declarations and definitions (T_2: type), instance declarations and concrete examples (T_3: instance) that support the proof.

Highest or goal layer: Represents the core theorem or proposition to be proved (T_4: goal), which relies on the information introduced in the lower layers.

These layers follow a natural partial order: $context \prec case \prec type \prec instance \prec goal$. Structuring mathematical reasoning within this hierarchy yields two key benefits:

- *Proper Scoping*: Contextual elements and definitions are confined to their appropriate levels. Intuitively, each concept is most meaningfully analyzed in conjunction with others at the same level, ensuring logical coherence and clarity.

- *Clear Semantic Flow*: The reasoning progresses seamlessly from foundational definitions to the final goal, reflecting the natural and intuitive structure of mathematical arguments.

3.2 Information Flow

We want to exploit the hierarchical structure by incorporating flow control into the model. Let T be the set of all tokens of the input theorem. We use t_i, t_j to denote individual tokens, L for the number of transformer layers, and $1 \leq l \leq L$ for layer indices. For tokens t_i, t_j in layer l, we define:

- $att_l(t_i, t_j)$: attention score from t_i to t_j, representing how much t_i will affect embedding of t_j at layer l,
- M_{ij}: binary attention mask, controlling the information flow from t_i to t_j,
- $\alpha_l = 1 - l/L$: layer-wise adaptation factor, which attenuates flow control for deeper layers.

We use level(t_i) to denote the hierarchical level of token t_i, taking value from $\{0, 1, 2, 3, 4\}$, corresponding to the five levels in our hierarchy. By controlling attention flow based on these levels, we encourage the model to follow natural mathematical reasoning patterns, where higher-level concepts build upon lower-level foundations.

4 Approach

To enhance the model's comprehension of the hierarchical structure and its ability to reason in alignment with it, we propose a two-step approach. First, we extract the flow pattern from the input by identifying different hierarchical levels in mathematical statements. Second, we guide the model's attention through a specialized loss function that encourages the model to respect these hierarchical relationships during training.

4.1 Extract Flow Pattern

In mathematical reasoning, different components of a statement naturally form a hierarchy. We identify five distinct levels (labeled 0 to 4): basic tokens, case-specific elements, type definitions, problem instances, and goal statements. The flow from token t_i to token t_j may follow one of three types, based on their hierarchical levels:

$$\begin{cases} \text{Unrestricted} & \text{if level}(t_i) = \text{level}(t_j) \\ \text{Guided} & \text{if level}(t_i) < \text{level}(t_j) \\ \text{Limited} & \text{if level}(t_i) > \text{level}(t_j) \end{cases} \quad (1)$$

While sharing similar algorithmic treatment, unrestricted and guided flows are distinctly categorized due to their different functional roles. Our experiments 5.3 show these flows develop distinct patterns during training: unrestricted flow shows reduced attention proportion, while guided flow demonstrates increased dominance. This structure ensures that semantic dependencies respect the hierarchical nature of mathematical reasoning, with tokens primarily attending to those at the same or lower levels, while limiting attention in the reverse direction to maintain logical consistency.

Algorithm 1: Hierarchical Attention Implementation

```
Input: Theorem text T, Model layers L
Output: Flow loss L_flow
/* Initialize hierarchical levels */
Parse input into level sets {T_0, ..., T_4} ;
 // Using Algorithm 2
;
Initialize attention mask M, L_flow ← 0;
for each layer l in 1 to L do
    α_l ← (1 − l/L) ;           // Layer adaptation factor
    for tokens t_i, t_j in input do
        /* Construct attention mask */
        if level(t_i) ≤ level(t_j) then
            M_ij ← 1 ;           // Allow upward/horizontal flow
        else
            M_ij ← 0 ;           // Limit downward flow
        /* Compute loss contribution */
        invalid_flow ←
          att_l(t_i, t_j) · (1 − M_ij);
        L_flow ←
          L_flow + α_l · ReLU(invalid_flow);
L_flow ← L_flow/|T|;
return L_flow;
```

4.2 Algorithm Implementation

Based on these flow patterns, we implement a hierarchical attention mechanism as shown in Algorithm 1. The algorithm first parses the input into different hierarchical levels using string pattern matching to identify key mathematical components. It then constructs attention masks and computes a flow loss that penalizes attention patterns violating hierarchical constraints.

The flow loss $\mathcal{L}_{flow}$ penalizes attention patterns that violate hierarchical constraints:

$$\mathcal{L}_{flow} = \frac{1}{|T|}\sum_{l=1}^{L}\alpha_l \cdot \sum_{i,j}\text{ReLU}(att_l(t_i,t_j)\cdot(1-M_{ij})) \tag{2}$$

where $\alpha_l = (1 - \frac{l}{L})$ provides stronger regularization in earlier layers while allowing more flexibility in later layers.

The final training objective combines this flow loss with the standard cross-entropy loss $\mathcal{L}_{LM}$:

$$\mathcal{L} = \mathcal{L}_{LM} + \lambda\mathcal{L}_{flow} \tag{3}$$

where λ controls the strength of hierarchical constraints. A larger λ enforces stricter adherence to the hierarchy, while a smaller value allows more flexible attention patterns.

In summary, our approach:

- Identifies natural hierarchical levels in mathematical statements.
- Guides attention patterns to respect hierarchical relationships.
- Enables flexible reasoning through layer-wise adaptation.

5 Experiments

In this section, we evaluate our method through comprehensive experiments on multiple theorem-proving benchmarks.

5.1 Experimental Setup

Training Data and Configuration We use LeanDojo Benchmark 4[*] as our training dataset. The training process involves fine-tuning a Pythia-2.8B[†] (Biderman et al., 2023) model for 3 epochs. Detailed hyperparameters and training configurations are provided in Appendix A.1.

Evaluation Protocol We conduct comprehensive evaluations across four benchmark datasets: miniF2F (test/valid)[‡] and ProofNet (test/valid)[§]. Our evaluation employs two complementary strategies: best-first search and single-pass sampling, to demonstrate the robustness of our method (detailed algorithms in Appendix A.2).

For both strategies, we define the computation budget as $K \times T$, where T indicates the number of expansion iterations, which is set to 100 across all our experiments, and $K = N \times S$. For the best-first search, N represents the number of parallel search attempts and S denotes the number of tactics generated per expansion. For single-pass sampling, N represents the total number of sampling attempts per problem, while S is fixed to 1 as only one tactic is attempted at each expanded node. The search process employs parallel sampling with fixed time constraints per theorem. In the following sections, we use K to denote the product of N and S for simplicity.

Our method is a general-purpose fine-tuning technique that can be applied to any formal theorem-proving system. For empirical validation, we chose LLMSTEP (Welleck and Saha, 2023) as our primary baseline, which provides full access to its model, dataset, and hyperparameters, ensuring complete reproducibility of our comparative analysis.

5.2 Main Results

We present a comparative analysis of our method against the baseline, highlighting its performance and advancements.

Metrics We evaluate our method using two key metrics: pass@K accuracy and proof complexity. The pass@K metric measures the model's ability to generate a valid proof within K sampling attempts, where $K = N \times S$ represents the total number of tactic samples considered during this iteration of proof search.

For proof conciseness analysis, we measure the number of proof steps required to solve the goals. Let $\mathcal{T}_{com}$ be the set of theorems successfully proved by both methods with different proof lengths. For each theorem $t \in \mathcal{T}_{com}$, we define its proof complexity as:

$$C(t, m) = |p_{t,m}| \tag{4}$$

where $p_{t,m}$ is the proof generated for theorem t using method m, and $|p_{t,m}|$ denotes the number of proof steps. We then compute the average complexity ratio:

$$R_{avg} = \frac{1}{|\mathcal{T}_{com}|}\sum_{t\in\mathcal{T}_{com}}\frac{C(t, \text{ours})}{C(t, \text{baseline})} \tag{5}$$

[*]Yang, K. (2023). LeanDojo Benchmark (v1) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.8016386

[†]https://huggingface.co/EleutherAI/pythia-2.8b

[‡]https://huggingface.co/datasets/cat-searcher/minif2f-lean4

[§]https://huggingface.co/datasets/UDACA/proofnet-lean4

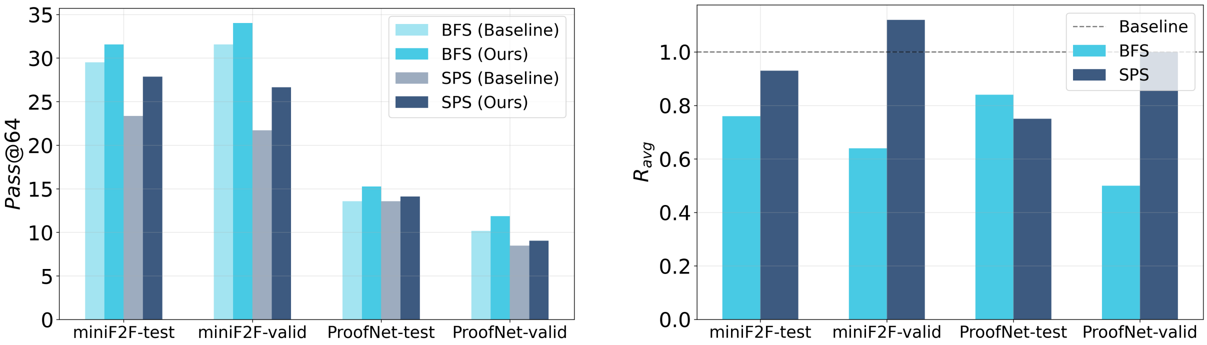

Figure 2: Performance comparison between our method and baseline at $K = 64$. **Left:** Pass rate comparison across miniF2F (test/valid) and ProofNet (test/valid) datasets. Best-first search (BFS) consistently outperforms single-pass sampling (SPS), with our method further enhancing BFS performance. Solid bars represent our method while transparent bars represent the baseline. **Right:** Proof complexity ratio (R_{avg}), where values below 1.0 (dashed line) indicate more concise proofs. Our method with BFS achieves consistent complexity reductions across all datasets.

Table 1: Results on miniF2F test set with best-first search strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
1	**14.75**	14.34	-	-	-	-
2	**18.44**	17.62	-	-	-	-
4	22.54	**23.36**	-	-	-	-
8	26.23	26.23	2.00	**1.86**	**0.93**	11.67
16	**29.10**	28.28	2.11	**1.50**	**0.71**	13.24
32	29.51	**31.15**	1.89	**1.67**	**0.88**	12.50
64	29.51	**31.56**	2.10	**1.60**	**0.76**	8.11

Table 2: Results on miniF2F validation set with best-first search strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
1	12.70	**13.52**	-	-	-	-
2	**15.16**	14.75	-	-	-	-
4	20.49	**23.77**	-	-	-	-
8	27.05	**29.51**	2.83	**2.67**	**0.94**	9.68
16	31.15	**33.20**	2.89	**1.89**	**0.65**	13.89
32	31.56	**34.02**	3.11	**2.00**	**0.64**	12.00
64	31.56	**34.02**	3.11	**2.00**	**0.64**	12.68

Table 3: Results on miniF2F test set with single-pass sampling strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
1	9.84	**18.44**	-	-	-	-
2	12.30	**20.90**	-	-	-	-
4	16.80	**24.18**	-	-	-	-
8	19.63	**25.00**	1.95	**1.86**	**0.95**	51.16
16	20.49	**26.23**	**1.85**	1.92	1.04	26.00
32	23.36	**26.64**	1.83	**1.78**	**0.97**	15.38
64	23.36	**27.87**	2.00	**1.85**	**0.93**	23.21

Table 4: Results on miniF2F validation set with single-pass sampling strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
1	9.43	**14.75**	-	-	-	-
2	12.30	**15.16**	-	-	-	-
4	16.80	**20.08**	-	-	-	-
8	18.03	**21.13**	2.33	**1.73**	**0.74**	37.50
16	18.44	**24.59**	1.95	**1.89**	**0.97**	43.18
32	20.08	**25.41**	1.92	1.92	1.00	27.66
64	21.72	**26.64**	**2.12**	2.38	1.12	16.33

This metric provides a direct measure of our method's proof conciseness, where $R_{avg} < 1$ indicates that our method generally produces shorter proofs. Note that we only consider theorems where **both methods succeed but generate proofs of different lengths**, as this provides a meaningful comparison of the proof conciseness. We also report Diff. (%), which indicates the percentage of such theorems among all theorems that both methods successfully prove, reflecting how often the methods differ in their proof strategies.

Overview Figure 2 presents a comprehensive evaluation of our method across miniF2F (test/valid) and ProofNet (test/valid) datasets at $K = 64$. The results demonstrate that best-first search (BFS) is the superior search strategy across all datasets, consistently outperforming single-pass sampling (SPS). When combined with our hierarchical attention mechanism, BFS achieves even stronger results. For example, on the miniF2F test set, our method improves the pass rate by 2.05% while reducing proof complexity by 23.81%. Similar improvements are observed on the ProofNet test set, with a 1.69% increase in pass rate and a 16.50% reduction in proof complexity. Notably, our method also significantly improves SPS performance, particularly on the miniF2F dataset where we observe pass rate improvements of 4.51% and 4.92% on test and valid sets respectively.

Results on miniF2F Tables 1-4 present comprehensive results on the miniF2F benchmark. With best-first search, our method achieves consistent improvements in pass rates at higher computation budgets, reaching 31.56% on test set (vs. baseline's 29.51%) and 34.02% on validation set (vs. baseline's 31.56%). The performance gain becomes more pronounced as the computation budget increases, particularly when K exceeds 16.

Single-pass sampling results also demonstrate the effectiveness of our method, achieving 27.87% and 26.64% pass rates on test and validation sets respectively at K = 64, compared to baseline's 23.36% and 21.72%. This represents substantial improvements of 4.51% and 4.92% respectively.

For proof conciseness evaluation, we focus on higher computation budget scenarios ($K \geq 8$) where sufficient successful proofs are available for reliable complexity comparison. At K = 64, our method demonstrates significant advantages in proof conciseness with the search strategy, reducing the average proof length from 3.11 to 2.00 steps (R_{avg} = 0.64) on the validation set and from 2.10 to 1.60 steps (R_{avg} = 0.76) on the test set. The reliability of these complexity metrics is supported by a substantial proportion of comparable cases (Diff.), where both methods succeed but with different proof lengths. For instance, at K = 64 with best-first search, these comparable cases constitute 8.11% and 12.68% of all successful proofs for test and validation sets respectively, providing a meaningful sample size for complexity comparison. Similar reliability is observed in single-pass sampling, where Diff. reaches 23.21% and 16.33%, ensuring the robustness of the reported complexity improvements.

Results on ProofNet Tables 5-8 present the results on ProofNet benchmark. With best-first search strategy, our method achieves consistent improvements in PASS rates at higher computation budgets, reaching 15.25% on test set (vs. baseline's 13.56%) and 11.86% on validation set (vs. baseline's 10.17%) at $K = 64$.

Single-pass sampling results also demonstrate the effectiveness of our method. On the test set, our method shows consistent improvements across computation budgets, achieving 14.12% at $K = 64$ compared to baseline's 13.56%, with improvements ranging from 1.70% to 2.63%. On the validation set, while performance is initially comparable at $K = 16$ (both 7.34%), our method shows improvement at higher computation budgets, reaching 9.04% at $K = 64$ compared to baseline's 8.47%.

For proof conciseness evaluation at $K = 64$, our method demonstrates significant advantages across all settings. With the search strategy, the average proof length decreases from 2.00 to 1.67 steps ($R_{avg} = 0.84$) on the test set and from 2.00 to 1.00 steps ($R_{avg} = 0.50$) on the validation set, based on 26.09% and 18.75% of differing proofs respectively. The single-pass sampling shows similar improvements with $R_{avg} = 0.75$ on the test set across 21.74% of differing cases.

Table 5: Results on ProofNet test set with best-first search strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
16	11.86	11.86	-	-	-	-
32	13.56	**14.69**	1.83	1.83	1.00	28.57
64	13.56	**15.25**	2.00	**1.67**	**0.84**	26.09

Table 6: Results on ProofNet validation set with best-first search strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
16	9.04	**10.73**	-	-	-	-
32	9.04	**10.73**	2.00	**1.00**	**0.50**	12.50
64	10.17	**11.86**	2.00	**1.00**	**0.50**	18.75

5.3 Visualization and Analysis of Attention Patterns

The attention distribution analysis shown in Figure 3 demonstrates that our mechanism successfully implements and maintains the designed information flow structure (Equation 1) throughout the model. Our analysis reveals several key findings across both constrained and unconstrained layers:

5.3.1 Implementation of Limited Flow Constraint

Our approach enforces the limited flow constraint by minimizing attention flows from higher to lower levels across all layers. In constrained layers (Figure 3, left), this is evidenced by the near-zero per-

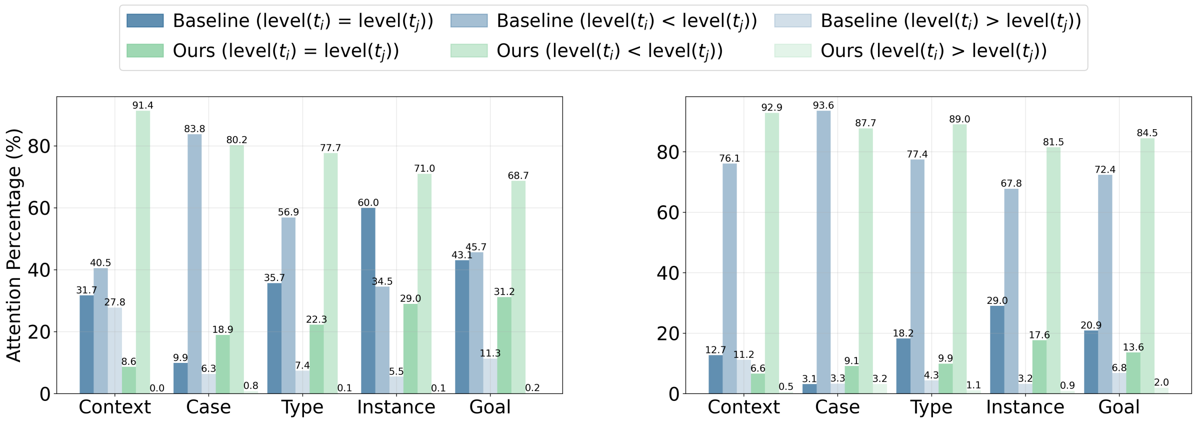

Figure 3: Attention distribution analysis in different layers. **Left:** Hierarchy-constrained layers (where $\alpha_l \neq 0$). **Right:** Unconstrained layers (where $\alpha_l = 0$). This visualization is derived from averaging attention patterns across all evaluation samples on the LeanDojo Benchmark 4 test set. The x-axis represents different hierarchical levels, while the y-axis shows the percentage of attention scores, combining both cases where the level's tokens serve as source (t_i) and target (t_j). Blue and green bars represent the baseline and our method respectively, with different transparency levels indicating different attention flow types based on the relationship between source level(t_i) and target level(t_j).

Table 7: Results on ProofNet test set with single-pass sampling strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
16	9.60	**11.30**	-	-	-	-
32	10.17	**12.80**	2.00	2.00	1.00	31.25
64	13.56	**14.12**	2.40	**1.80**	**0.75**	21.74

Table 8: Results on ProofNet validation set with single-pass sampling strategy.

K	PASS(%)		COMPLEXITY		R_{avg}	Diff. (%)
	Baseline	Ours	Baseline	Ours		
16	7.34	7.34	-	-	-	-
32	**8.47**	7.34	1.50	1.50	1.00	18.18
64	8.47	**9.04**	1.50	1.50	1.00	30.77

centages of level(t_i) $>$ level(t_j) attention across all hierarchical levels, compared to the baseline's substantial invalid flows ranging from 5.5% to 27.8%. Remarkably, this pattern persists in unconstrained layers (Figure 3, right), where invalid flows remain minimal (ranging from 0.5% to 3.2% across different levels), demonstrating the robustness of our hierarchical structure.

5.3.2 Effectiveness of Guided Flow Design

Our method successfully implements and maintains the guided flow design throughout the model. In constrained layers, the goal level effectively integrates information from lower levels with 68.7% upward attention while restricting reverse flows to just 0.2%. Type and instance levels receive substantial guided information flow from lower levels (77.7% and 71.0% respectively), demonstrating strong hierarchical information propagation. This pattern strengthens in unconstrained layers, where the goal level receives even stronger attention from lower levels (84.5%), and type and instance levels maintain robust upward attention flows (89.0% and 81.5% respectively).

5.3.3 Global Impact on Model Behavior

The consistency of hierarchical patterns between constrained and unconstrained layers is particularly significant, indicating that our method induces a global, coherent hierarchical information processing framework. Rather than merely responding to external constraints, the model appears to have internalized the hierarchical structure, as evidenced by the preservation of desired attention patterns in unconstrained layers. This seamless continuation of attention patterns throughout the model architecture suggests that our hierarchical attention mechanism effectively shapes the model's overall information processing strategy, establishing a stable and consistent hierarchical flow structure.

6 Conclusion

We introduced Hierarchical Attention, a regularization method that aligns transformer attention with mathematical reasoning structures through a five-level hierarchy. Our approach balances structured information flow with the flexibility needed for complex proofs through layer-wise adaptation. Experimental results show improved proof success rates and conciseness across multiple benchmarks, while attention pattern analysis confirms the method's effectiveness in helping models internalize mathematical hierarchies. The consistent improvements demonstrate a promising direction for bridging neural language models and mathematical reasoning.

Limitations

Our approach has three main limitations: (1) the hierarchy definition is specific to Lean's semantics and may require adaptation for other proof languages, (2) the fixed hierarchy structure may limit dynamic reasoning patterns, and (3) data constraints prevented evaluation on advanced models like DeepSeek-Prover (Xin et al., 2024) and InternLM-Math (Ying et al., 2024b). Future work could explore adaptive hierarchies and the cross-domain generalization.

Ethical Considerations

Our work focuses on improving theorem proving through Hierarchical Attention while addressing several ethical considerations. We use publicly available datasets, including LeanDojo Benchmark 4 under the MIT license[¶], and strictly follow data usage policies. While mathematical content is generally objective, we acknowledge potential biases in theorem selection and proof styles. Our method, though designed for positive applications, should be used with human oversight as it could potentially generate misleading proofs. To promote transparency and reproducibility, we will release our code and models with appropriate licenses and usage guidelines.

References

Ibrahim Abdelaziz, Maxwell Crouse, Bassem Makni, Vernon Austel, Cristina Cornelio, Shajith Ikbal, Pavan Kapanipathi, Ndivhuwo Makondo, Kavitha Srinivas, Michael Witbrock, et al. 2022. Learning to guide a saturation-based theorem prover. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):738–751.

[¶] https://github.com/lean-dojo/LeanDojo/blob/main/LICENSE

Eser Aygün, Zafarali Ahmed, Ankit Anand, Vlad Firoiu, Xavier Glorot, Laurent Orseau, Doina Precup, and Shibl Mourad. 2020. Learning to prove from synthetic theorems. *arXiv preprint arXiv:2006.11259*.

Eser Aygün, Ankit Anand, Laurent Orseau, Xavier Glorot, Stephen M Mcaleer, Vlad Firoiu, Lei M Zhang, Doina Precup, and Shibl Mourad. 2022. Proving theorems using incremental learning and hindsight experience replay. In *International Conference on Machine Learning*, pages 1198–1210. PMLR.

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

Andrej Bauer, Matej Petković, and Ljupco Todorovski. 2024. Mlfmf: data sets for machine learning for mathematical formalization. *Advances in Neural Information Processing Systems*, 36.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. 2000. A deductive database approach to automated geometry theorem proving and discovering. *Journal of Automated Reasoning*, 25(3):219–246.

Karel Chvalovskỳ, Konstantin Korovin, Jelle Piepenbrock, and Josef Urban. 2023. Guiding an instantiation prover with graph neural networks. In *LPAR*, pages 112–123.

Maxwell Crouse, Ibrahim Abdelaziz, Bassem Makni, Spencer Whitehead, Cristina Cornelio, Pavan Kapanipathi, Kavitha Srinivas, Veronika Thost, Michael Witbrock, and Achille Fokoue. 2021. A deep reinforcement learning approach to first-order logic theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6279–6287.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. 2015. 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*, pages 378–388. Springer.

Kefan Dong and Tengyu Ma. 2025. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *Preprint*, arXiv:2502.00212.

Niklas Eén and Niklas Sörensson. 2003. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer.

Deborah Ferreira and André Freitas. 2020a. Natural language premise selection: Finding supporting statements for mathematical text. *arXiv preprint arXiv:2004.14959*.

Deborah Ferreira and André Freitas. 2020b. Premise selection in natural language mathematical texts. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7365–7374.

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

Achille Fokoue, Ibrahim Abdelaziz, Maxwell Crouse, Shajith Ikbal, Akihiro Kishimoto, Guilherme Lima, Ndivhuwo Makondo, and Radu Marinescu. 2023. An ensemble approach for automated theorem proving based on efficient name invariant graph neural representations. *arXiv preprint arXiv:2305.08676*.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. 2021. Proof artifact cotraining for theorem proving with language models. *arXiv preprint arXiv:2102.06203*.

Edvard K Holden and Konstantin Korovin. 2025. Graph sequence learning for premise selection. *Journal of Symbolic Computation*, 128:102376.

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. 2018. Gamepad: A learning environment for theorem proving. *arXiv preprint arXiv:1806.00608*.

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

Albert Q Jiang, Wenda Li, and Mateja Jamnik. 2023. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022a. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. 2021. Lisa: Language models of isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pages 378–392.

Albert Qiaochu Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. 2022b. Thor: Wielding hammers to integrate language models and automated theorem provers. *Advances in Neural Information Processing Systems*, 35:8360–8373.

Laura Kovács and Andrei Voronkov. 2013. First-order theorem proving and vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer.

Andrzej Stanisław Kucik and Konstantin Korovin. 2018. Premise selection with neural networks and distributed representation of features. *arXiv preprint arXiv:1807.10268*.

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

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. 2024. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*.

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

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, et al. 2023. Fimo: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*.

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

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, et al. 2024. Process-driven autoformalization in lean 4. *arXiv preprint arXiv:2406.01940*.

Jack McKeown and Geoff Sutcliffe. 2023. Reinforcement learning for guiding the e theorem prover. In *The International FLAIRS Conference Proceedings*, volume 36.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. 2023. Magnushammer: A transformer-based approach to premise selection. *arXiv preprint arXiv:2303.04488*.

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

Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. Autoformalizing euclidean geometry. *arXiv preprint arXiv:2405.17216*.

Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. 2020. Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974.

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

Kebin Peng and Dianfu Ma. 2017. Tree-structure cnn for automated theorem proving. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*, pages 3–12. Springer.

Bartosz Piotrowski and Josef Urban. 2020. Stateful premise selection by recurrent neural networks. *arXiv preprint arXiv:2004.08212*.

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

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

Michael Rawson and Giles Reger. 2019. A neurally-guided, parallel theorem prover. In *Frontiers of Combining Systems: 12th International Symposium, FroCoS 2019, London, UK, September 4-6, 2019, Proceedings 12*, pages 40–56. Springer.

Michael Rawson and Giles Reger. 2020. Directed graph networks for logical reasoning. In *PAAR+ SC^2@ IJCAR*, pages 109–119.

Michael Rawson and Giles Reger. 2021. lazycop: Lazy paramodulation meets neurally guided search. In *Automated Reasoning with Analytic Tableaux and Related Methods: 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6–9, 2021, Proceedings 30*, pages 187–199. Springer.

Jason Rute, Miroslav Olšák, Lasse Blaauwbroek, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. 2024. Graph2tac: learning hierarchical representations of math concepts in theorem proving. *arXiv preprint arXiv:2401.02949*.

Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 1–10.

Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. 2023. Passport: Improving automated formal verification using identifiers. *ACM Transactions on Programming Languages and Systems*, 45(2):1–30.

Stephan Schulz. 2002. E–a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126.

Martin Suda. 2021. Improving enigma-style clause selection while learning from history. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 543–561. Springer.

The Coq Development Team. 2024. *Coq*. URL https://coq.inria.fr.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. 2025a. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *Preprint*, arXiv:2504.11354.

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

Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. 2023a. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, et al. 2023b. 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)*, pages 12632–12646.

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

Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef Urban. 2020. Exploration of neural machine translation in autoformalization of mathematics in mizar. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 85–98.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. 2018. First experiments with neural translation of informal to formal mathematics. In *Intelligent Computer Mathematics: 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings 11*, pages 255–270. Springer.

Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. 2025b. Ma-lot: Multi-agent lean-based long chain-of-thought reasoning enhances formal theorem proving. *Preprint*, arXiv:2503.03205.

Sean Welleck and Rahul Saha. 2023. Llmstep: Llm proofstep suggestions in lean. *arXiv preprint arXiv:2310.18457*.

Qinzhuo Wu, Qi Zhang, and Xuanjing Huang. 2022a. Automatic math word problem generation with topic-expression co-attention mechanism and reinforcement learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:1061–1072.

Yuhuai Wu. 2022. Formal premise selection with language models. In *Conference on Artificial Intelligence and Theorem Proving (AITP)*, volume 4.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022b. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368.

Zijian Wu, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Lean-github: Compiling github lean repositories for a versatile lean prover. *arXiv preprint arXiv:2407.17227*.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. 2025. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *Preprint*, arXiv:2502.03438.

Kaiyu Yang and Jia Deng. 2019. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, pages 6984–6994. PMLR.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2024. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024a. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. 2024b. Internlm-math: Open math large language models toward verifiable reasoning. *Preprint*, arXiv:2402.06332.

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

Xueliang Zhao, Wenda Li, and Lingpeng Kong. 2023. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*.

A Appendix

A.1 Training Details

We use Pythia-2.8B[‖] as our base model. The training data is sourced from LeanDojo Benchmark 4 [**], which consists of 169,530 samples for training and 3,606 samples for validation.

We train the model for 3 epochs on 8 NVIDIA A800 GPUs using DeepSpeed[††] with ZeRO-3 optimization, taking approximately 40 hours. The training uses a per-device batch size of 2 with gradient accumulation steps of 2, resulting in an effective batch size of 32. We adopt a learning rate of 1e-5 with a cosine decay schedule and 3% warmup ratio. The training process employs FP16 precision without weight decay, and ZeRO-3 is configured with parameter and optimizer state partitioning across GPUs. For reproducibility, we set the random seed to 42 across all experiments.

During training, we evaluate the model every 500 steps and save checkpoints at the same frequency, maintaining the 3 most recent checkpoints. The best model is selected based on validation performance at the end of training. The training objective combines the standard cross-entropy loss with our hierarchical flow loss. Table 9 shows the specific hyperparameters (λ and L) used for different evaluation sets.

Table 9: Hyperparameters for different evaluation sets.

Dataset	λ	L
miniF2F (test)	0.1	4
miniF2F (valid)	0.1	16
ProofNet (test)	0.2	16
ProofNet (valid)	0.2	4

A.2 Evaluation algorithm

We implement two evaluation algorithms for theorem proving: best-first search and single-pass sampling. Both algorithms share the same computation budget $K \times T$, where $T = 100$ is the maximum expansion steps.

Best-First Search maintains a priority queue of states ranked by trajectory score $\sum_{j=0}^{i-1} \log p(a_j|s_j)$. For each expansion, it selects the highest-scoring state s_i, generates S candidate tactics, and creates new states by applying valid tactics. The search succeeds when reaching a state with no remaining goals within N expansions.

Single-Pass Sampling runs K parallel proof attempts. Each attempt samples tactics sequentially until finding a valid one or reaching the attempt limit. A proof succeeds if it completes within N valid tactics. This approach simplifies the search process by setting $S = 1$ and focusing on trajectory sampling rather than state ranking.

A.3 Ablation Studies

A.3.1 Layer-wise Adaptation Mechanism

To validate the effectiveness of our layer-wise adaptation mechanism ($\alpha_l = 1 - l/L$), we conduct ablation studies on miniF2F and ProofNet benchmarks using best-first search with $K = 64$. The results are shown in Table 10.

Table 10: Ablation study results on miniF2F and ProofNet benchmarks with best-first search ($K = 64$).

Method	miniF2F		ProofNet	
	Test	Valid	Test	Valid
PASS (baseline)	29.51	31.56	13.56	10.17
PASS (w/o adaptation)	30.74	32.34	14.69	11.30
PASS (w/ adaptation)	**31.56**	**34.02**	**15.25**	**11.86**
R_{avg} (w/o adaptation)	**0.53**	0.82	**0.69**	**0.50**
R_{avg} (w/ adaptation)	0.76	**0.64**	0.84	**0.50**
Diff.(w/o adaptation) (%)	9.86	11.27	22.73	18.75
Diff.(w/ adaptation) (%)	8.11	12.68	26.09	18.75

The results demonstrate an interesting trade-off in our layer-wise adaptation mechanism. Without adaptation, where hierarchical constraints are applied uniformly across layers, the model achieves better proof complexity ratios across three benchmarks but lower pass rates. This suggests that gradually reducing the constraint strength in deeper layers through layer-wise adaptation ($\alpha_l = 1 - l/L$) helps achieve better proof success rates at the cost of slightly longer proofs. The superior pass rates across all benchmarks validate that our adaptive approach effectively enhances the model's theorem proving capabilities while maintaining reasonable proof complexity. Notably, even without layer-wise adaptation, our hierarchical attention mechanism still outperforms the baseline substantially in both pass rates and proof complexity, demonstrating the effectiveness of our basic hierarchical structure design.

[‖] https://huggingface.co/EleutherAI/pythia-2.8b

[**] Yang, K. (2023). LeanDojo Benchmark (v1) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.8016386

[††] https://github.com/microsoft/DeepSpeed

A.3.2 Hierarchical Structure Variants

To explore the impact of different hierarchical structures, we conducted ablation experiments comparing the fine-grained structure (with all five levels $T_0 - T_4$) against a coarse-grained variant where levels $T_1 - T_3$ are merged into a single level. Table 11 shows the results with best-first search ($K = 64$). On the miniF2F benchmark, the coarse-grained variant achieves better proof complexity ratios (R_{avg}) on the test set (0.67 vs 0.76), while the fine-grained structure achieves better complexity on the validation set (0.64 vs 0.82). More importantly, the fine-grained approach delivers superior pass rates across both test and validation sets. Similar patterns are observed in the ProofNet benchmark. These results demonstrate that distinguishing between different reasoning elements (case analysis, type declarations, and instances) is beneficial for overall theorem-proving performance, with mixed effects on proof complexity. The fine-grained structure provides the model with more detailed information about the relationships between different elements in the proof state, enabling more accurate reasoning and higher success rates.

Table 11: Comparison of hierarchical structure variants on miniF2F and ProofNet benchmarks.

Method	miniF2F		ProofNet	
	Test	Valid	Test	Valid
PASS (baseline)	29.51	31.56	13.56	10.17
PASS (coarse-grained)	30.33	32.38	14.12	11.30
PASS (fine-grained)	**31.56**	**34.02**	**15.25**	**11.86**
R_{avg} (coarse-grained)	**0.67**	0.82	**0.73**	1.26
R_{avg} (fine-grained)	0.76	**0.64**	0.84	**0.50**
Diff.(coarse-grained) (%)	8.33	11.43	26.32	21.43
Diff.(fine-grained) (%)	8.11	12.68	26.09	18.75

A.4 Explicit Level Tags Baseline

To further evaluate the effectiveness of different structural representation methods, we implemented a baseline that uses explicit level tags. This experiment was designed to compare two fundamentally different approaches to representing structure: directly adding explicit tags in the input data versus implicitly guiding hierarchical information flow through attention mechanisms. In this baseline, we modified the input format to include explicit tags indicating the hierarchical level of each component:

```
{
    Input: "<context>...</context>...
    <type>...</type>...
    <instance>...</instance>...
    <goal>...</goal>...",
}
```

Table 12 shows the results with best-first search ($K = 64$). The explicit tags approach significantly underperformed both the baseline and our method across all datasets. These results indicate that simply annotating data with explicit level tags is ineffective and potentially detrimental for theorem proving.

Table 12: Comparison between baseline, explicit tags approach, and our method with best-first search ($K = 64$).

Method	miniF2F		ProofNet	
	Test	Valid	Test	Valid
PASS (baseline)	29.51	31.56	13.56	10.17
PASS (explicit tags)	21.88	16.80	9.04	7.34
PASS (ours)	**31.56**	**34.02**	**15.25**	**11.86**
R_{avg} (explicit tags)	2.01	1.78	1.30	2.00
R_{avg} (ours)	**0.76**	**0.64**	**0.84**	**0.50**
Diff.(explicit tags) (%)	24.44	17.50	40.00	18.18
Diff.(ours) (%)	8.11	12.68	26.09	18.75

A.5 Input Parsing Algorithm

To implement our hierarchical structure, we developed a rule-based parsing algorithm that identifies different structural components in theorem text, as shown in Algorithm 2. This lightweight pattern-matching approach identifies key mathematical components through syntactic indicators: goal statements (containing turnstile $\vdash$), case analysis statements (starting with "case"), type declarations (containing "Type" and colon), and instance definitions (containing colon but neither "Type" nor turnstile). The parser maintains context across multi-line statements by inheriting the level of previous lines when appropriate, ensuring accurate hierarchical structure capture with minimal computational overhead.

A.6 Case Studies

To demonstrate the effectiveness of our hierarchical attention mechanism in generating concise proofs, we present three representative examples from different mathematical domains in the miniF2F dataset.

These examples showcase how our hierarchical attention mechanism improves proof generation across different mathematical domains. In Table 13,

Algorithm 2: Hierarchical Structure Parsing

Input: Theorem text
Output: Hierarchical level information
Initialize empty segments list;

```
/* Initialize context level */
Add context tokens to segments with level context;
Extract Lean4 code block between context;
current ← context;
/* Process text line by line */
for each line in Lean4 code do
    /* Determine hierarchical level */
    if line contains ⊢ then
        current ← goal;
    else if line starts with case then
        current ← case;
    else if line contains Type and : then
        current ← type;
    else if line contains : but not Type or ⊢ then
        current ← instance;
    else
        Maintain current for continued lines;
    Record segment with position and level;
return Hierarchical level information;
```

our model directly combines the function definition with the given value, eliminating the need for intermediate expansion. Table 14 demonstrates improved pattern recognition, where our model directly applies the appropriate modular multiplication rule instead of decomposing the operation into addition. Table 15 shows enhanced tactic understanding, combining function expansion with field simplification in a single step. The consistent reduction in proof steps across these diverse examples demonstrates how our hierarchical attention mechanism enables better mathematical reasoning.

Lean4 Statement

```
theorem mathd_algebra_148 (c : Real) (f : Real -> Real)
  (h0 : ∀ x, f x = c * x^3 - 9 * x + 3)
  (h1 : f 2 = 9) : c = 3
```

Baseline Proof

```
rw [h0] at h1    -- Substitute f(2) with its definition
linarith         -- Solve the resulting equation c * 8 - 18 + 3 = 9
```

Our Proof

```
linarith only [h0 2, h1]  -- Directly solve using h0 applied to 2 and h1
```

Table 13: Case Study 1: Basic Algebra Problem

Lean4 Statement

```
theorem mathd_numbertheory_185 (n : Nat)
  (h0 : n % 5 = 3) : 2 * n % 5 = 1
```

Baseline Proof

```
rw [two_mul]          -- Convert 2 * n to n + n
rw [Nat.add_mod, h0]  -- Apply modular addition: (3 + 3) % 5 = 1
```

Our Proof

```
rw [Nat.mul_mod, h0]  -- Apply modular multiplication: 2 * 3 % 5 = 1
```

Table 14: Case Study 2: Number Theory Problem

Lean4 Statement

```
theorem amc12a_2016_p3 (f : Real -> Real -> Real)
  (h0 :  ∀ x,  ∀ (y) (_ : y != 0),
        f x y = x - y * Int.floor (x / y)) :
  f (3/8) (-(2/5)) = -(1/40)
```

Baseline Proof

```
simp [h0]                 -- Expand function definition
field_simp [two_ne_zero]  -- Simplify rational expressions
norm_cast                 -- Convert between types
```

Our Proof

```
field_simp [h0]  -- Combine function expansion and field simplification
norm_cast          -- Convert between types
```

Table 15: Case Study 3: Advanced Algebra Problem