Can Large Language Models Learn Formal Logic? A Data-Driven Training and Evaluation Framework

Yuan Xia*
University of Southern California
Los Angeles, CA, USA
yuanxia@usc.edu

Akanksha Atrey
Nokia Bell Labs
Murray Hill, NJ, USA
akanksha.atrey@nokia-bell-labs.com

Fadoua Khmaissia
Nokia Bell Labs
Murray Hill, NJ, USA
fadoua.khmaissia@nokia-bell-labs.com

Kedar S. Namjoshi
Nokia Bell Labs
Murray Hill, NJ, USA
kedar.namjoshi@nokia-bell-labs.com

Abstract

This paper studies the logical reasoning capabilities of language models. In this setting, the model takes assumptions and a goal as input and generates a formal proof deriving the goal from the assumptions. Incorrect proofs are caught by automated validation. We focus on the conceptually simple but technically challenging task of constructing proofs in Boolean logic, where it is feasible to train and test at scale with synthetic data. We apply randomization to generate a large, diverse corpus of proofs for training and testing. We introduce Template Transformation, a proof augmentation technique that generates syntactically varied proofs that share a common logical structure. We propose natural black-box tests to evaluate reasoning ability. By those measures, experiments show that LoRA-based fine tuning combined with Template Transformation significantly improves reasoning ability; however, reasoning accuracy declines sharply with increasing proof complexity. These findings highlight the need for fresh approaches to the reasoning question.

1 Introduction

In July 2024, The New York Times published an article aptly titled "*A.I. Can Write Poetry, but It Struggles With Math.*" [Lohr, 2024]. General Large Language Models (LLMs) are indeed excellent at manipulating text but can be poor at logical reasoning, which is at the heart of mathematics and computing. This work aims to investigate experimentally whether that is an inherent limitation.

LLM-based reasoning falls into two broad categories. Neuro-symbolic systems, such as AlphaProof [AlphaProof and teams, 2024], use LLMs to guide a proof-search engine, replacing hard-coded heuristics or human-provided hints. In the second category are LLMs that generate proof objects, such as Chain-of-Thought (CoT) [Wei et al., 2022]. (More related work in Appendix E.) While easy to follow, the natural-language CoT explanations are challenging to validate.

Instead, we choose to work entirely in a formal language. A "logic LLM" receives as input assumptions and a goal, both expressed in formal logic. The LLM produces a formal proof explaining how the goal follows from the assumptions. Incorrect proofs are caught by a proof checker.

We focus on reasoning in Boolean logic. Several considerations informed this choice. Boolean logic is the foundation for other logics (e.g., first order, modal, temporal, and the like). Therefore,

*Corresponding author: yuanxia@usc.edu

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

if Boolean reasoning turns out to be a challenging task for LLMs, it is unlikely that LLMs will do better on more sophisticated logics. Boolean logic has a compact proof system that simplifies synthetic proof generation for training and for the automatic validation of proof claims produced by an LLM. On the other hand, Boolean inference is inherently difficult, being the prototypical co-NP-complete question. Boolean reasoning may thus be viewed as the *C.elegans* [Brenner, 2002] of LLM reasoning: syntactically simple, semantically challenging, and exhibiting in the small many of the central difficulties of general reasoning.

The central issues are (1) a scarcity of real-world training data, i.e., valid proofs, and (2) a means of determining whether a black-box model is reasoning rather than memorizing or guessing. We address issue (1) by designing efficient randomized algorithms to synthesize valid proofs and through *Template Transformation*, a new data augmentation technique that aims to enhance generalization. We address issue (2) by designing meaningful, fully automated, black-box tests of reasoning ability. We train and evaluate on synthetically generated problems, with randomized large-scale testing that does not rely on curated benchmarks. Using our own formal syntax ensures that models have no prior exposure and enables automated proof validation, avoiding the ambiguities of natural language.

Initial experiments show that pre-trained instruct models have weak reasoning ability. Using only 9000 synthetically generated proofs as training data, our fine-tuned Llama-8B model reaches 0.98 accuracy on depth-7 proofs, outperforming GPT-4o's few-shot learning performance (0.7 accuracy), despite having far fewer parameters. Ablation studies show that larger models improve generalization to deeper proofs, while proof augmentation enhances performance with complex cases, especially for Llama-1B. As proof complexity increases, model accuracy drops sharply, likely due to limits in LLM context or the inherent problem difficulty. This abrupt decline, also observed in Boolean satisfiability tasks [Pan et al., 2025], highlights the need for new approaches to robust LLM reasoning.

2 Proofs and Proof Generation

There are many proof systems for Boolean logic. We use the well known Hilbert style of proof system [Mendelson, 2024] for its simplicity and compactness. We restrict attention to a sub-logic with implication ($\rightarrow$) as its only connective. Every formula of this type is satisfiable, but validity is co-NP-hard (see Appendix A.3). The proof system has only one inference rule, called Modus Ponens (MP), and two axiom schemas:

$$\begin{aligned} &A1 :: \quad (A \rightarrow (B \rightarrow A)) \\ &A2 :: \quad (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \end{aligned}$$

Colloquially, the first schema says that if both A and B hold, then A holds. The second schema says that if A and B together imply C and A is stronger than B, then A implies C by itself. The MP inference rule allows one to deduce B from the hypotheses "$A \rightarrow B$" and "A."

A *proof* is a sequence of steps, where each step contains a formula with a *justification*. A justification asserts that the formula is either (1) an assumption; or (2) is obtained by substitution into an axiom schema; or (3) is derived by applying modus ponens to two formulas at *earlier* steps of the proof. A proof is *valid* if each justification is correct and the goal formula appears at some proof step.

Example Proof [Transitivity]. Does the goal $p \rightarrow r$ follow from assumptions $p \rightarrow q$ and $q \rightarrow r$? This is just transitivity, which can be proved as follows. Our aim is to train an LLM to produce this (or equivalent) proof as output given the assumption formulas and the goal formula as input.

1. $(q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$, by Axiom A1 with substitution $A = q \rightarrow r, B = p$.
2. $q \rightarrow r$, by Assumption 2.
3. $p \rightarrow (q \rightarrow r)$, by MP on steps $1, 2$.
4. $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$, by Axiom A2 with substitution $A = p, B = q, C = r$.
5. $(p \rightarrow q) \rightarrow (p \rightarrow r)$, by MP on steps $4, 3$.
6. $p \rightarrow q$, by Assumption 1.
7. $p \rightarrow r$, by MP on steps $5, 6$.

We use a simple and efficient randomized backward (i.e., goal-directed) proof generation process to synthesize valid proofs of arbitrary size and depth. The process starts with a randomly chosen goal and generates a proof tree rooted at that goal. Some leaves of this tree are considered "open" nodes; they represent the assumptions under which the goal holds. All interior nodes record applications of modus ponens. Initially, the root node is open. An open leaf node labeled with formula g may be turned into an interior node by choosing a formula f at random, re-labeling this node as a modus ponens derivation of g from $f \to g$ and f, and placing those formulas on open leaf child nodes. An open leaf may also be "closed" if it can be obtained from an axiom schema through substitution. A proof in linear form is obtained by linearizing the proof tree.

An important observation is that a proof may also be viewed as a proof schema, to which one can apply arbitrary substitutions to obtain other valid proofs. For instance, say we uniformly replace q with y_1, and p with $y_1 \to y_2$, and r with $y_2 \to y_1$ in the transitivity proof. This creates a valid proof showing that the transformed goal $(y_1 \to y_2) \to (y_2 \to y_1)$ follows (by transitivity) from the transformed assumptions $(y_1 \to y_2) \to y_1$ and $y_1 \to (y_2 \to y_1)$.

Applying this replacement property, one can generate a large collection of proofs that are syntactically distinct but share the same proof structure. We call this technique *template transformation*. It is a form of data augmentation: by introducing variations on a common proof structure, LLMs are trained to focus on the deeper reasoning pattern instead of superficial syntax, like variable names. This is analogous to training enhancement for computer vision, where images undergo spatial transformations so models learn essential object features.

3 Evaluation

It is a challenge to distinguish reasoning from memorization, especially for large, black-box LLMs. We aim to do so with the following metrics, which assess the models' ability to handle fresh, unseen problems of varying degrees of complexity. (Further details are in Appendix C and D.)

- (Favoring Semantics over Syntax) If an LLM can solve a reasoning problem defined over variables p, q, it should solve the template-transformed problem where p, q are replaced by formulas f, g, respectively. This would indicate that the LLM recognizes the underlying abstract reasoning pattern.
- (Generalizing to Deeper and Wider Proofs) A reasoning LLM should discover proofs that are longer than those present in its training set. It should be able to solve "out of distribution" questions formulated over an expanded set of variables. Both abilities suggest that the LLM is not merely memorizing proofs up to a certain complexity.
- (Allowing Proof Diversity) A logical puzzle may have several valid proofs. We evaluate the accuracy of LLM-responses on semantic proof validity rather than on token-level syntactic equality. A generated valid proof that is not in the test set can indicate a level of creativity.

Training We fine-tune Llama3 8B and 1B models [Dubey et al., 2024] on synthetic Hilbert proofs, using 4-bit quantization and LoRA [Hu et al., 2021]. Training data consists of balanced proofs with depths 7, 10, and 13. Template Transformation is applied with probability $\alpha_{TT} = 0.7$, replacing variables with randomly generated expressions (up to 4 variables, max recursion depth 4). Convergence is determined through early stopping using exact match accuracy.

Testing We generate separate proof trees for evaluation to prevent data leakage, ensuring all formulas are unique. The validation set has 375 samples from training depths (125 each). For generalization, we use 450 test samples across depths 4-28 (50 per depth) and 400 samples across four expression widths (100 per width). Testing accuracy is the ratio of proofs passing our formal validator.

Experimental Results We evaluate our approach against GPT-4o (100B+ parameters) and base Llama models on training depths 7,10,13 (Table 1). For GPT-4o and pre-trained models, we test few-shot learning with 1-9 demonstrations, averaging results across settings. GPT-4o achieves 0.67 ± 0.08 accuracy at depth-7, degrading to 0.26 ± 0.04 at depth-13. Pre-trained Llama models fail completely ($\approx$0.00 accuracy). Our fine-tuned 8B model with Template Transformation achieves 0.98 accuracy at depth-7, significantly outperforming GPT-4o despite fewer parameters. The 1B model shows modest improvements (0.52 at depth-7), suggesting reasoning capabilities scale with model size. (Please refer to Appendix D for more details.)

Table 1: Proof Generation Accuracy on In-Distribution Test Data: Comparison between fine-tuned Llama models (trained on 9000 examples), GPT-4o few-shot learning (averaged across 1,3,6,9-shot learning settings), and pre-trained baseline models. Testing performed on proof depths (d) matching training distribution. w/ TT and w/o TT denote models fine-tuned with and without Template Transformation respectively.

	Accuracy by Proof Depth		
Model	$d = 7$	$d = 10$	$d = 13$
SOTA Performance			
GPT-4o (X-shot)	0.67±0.08	0.48±0.08	0.26±0.04
Pre-trained Base Models			
Llama-3.2-1B (X-shot)	≈ 0.00	≈ 0.00	≈ 0.00
Llama-3.1-8B (X-shot)	≈ 0.00	≈ 0.00	≈ 0.00
Our Approach - Small Model (Llama-3.2-1B)			
Fine-tuned w/o TT	0.52	0.30	0.02
Fine-tuned w/ TT	0.54	0.30	0.04
Our Approach - Large Model (Llama-3.1-8B)			
Fine-tuned w/o TT	0.78	0.40	0.28
Fine-tuned w/ TT	**0.98**	**0.66**	**0.32**

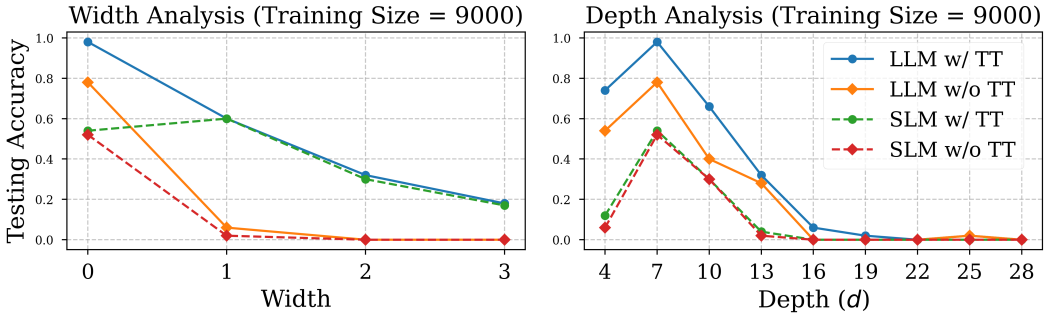

Figure 1: Comparison of Llama3-8B (LLM) and Llama3-1B (SLM) model variants' testing accuracies across different training data sizes. Left: Impact of proof width on accuracy. Right: Impact of proof depth on accuracy. Models with and without Template Transformation (TT) are compared.

We conduct ablation studies to systematically analyze how different factors affect our fine-tuned LLMs' performance, and to understand the limits of our proposed approach. The central results are shown in Figure 1. In summary, performance degrades with increasing proof complexity; initially, the degradation is smooth, but it is followed by a sharp drop. Template Transformation significantly improves handling of complex expressions. We also find that larger models show better resilience. The 8B model maintains reasonable accuracy up to depth 13, while the 1B model's performance drops more sharply after depth 10 (Please refer to Appendix D.3 for the full ablation study).

4 Discussion

Our results suggest that LLMs can be trained to develop generalizable capabilities for formal reasoning, up to a limit. Fine-tuning a small Llama model with a small synthetically generated proof set, enhanced with template transformations, boosts reasoning ability, surpassing the performance that significantly larger models such as GPT-4o attain with few-shot learning. However, there are limits, indicated by the sharp drop in accuracy on the proof-depth metric. While the precise cause is unclear, it does not necessarily indicate an inability to reason. The drop may be caused by LLM context limits, which results in early deductions being "forgotten," or it may be a manifestation of the inherent hardness of Boolean reasoning.

What the findings do indicate is the need for further research to understand the theoretical and practical limits of LLM reasoning ability. Everyday human reasoning goes well beyond basic propositional reasoning, incorporating quantification (e.g., "for all"), time (e.g., "yesterday") and mode (e.g., "possible"). "Commonsense" reasoning can be non-monotonic, allowing prior conclusions to be revised based on fresh facts [Strasser and Antonelli, 2024]. These aspects can all be studied within the Boolean setting, which forms a convenient, precise, and fully automatable "test tube" in which to systematically evaluate new approaches to the reasoning question.

References

AlphaProof and AlphaGeometry teams. AI achieves silver-medal standard solving International Mathematical Olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, July 2024.

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. In Raquel Fernández and Tal Linzen, editors, *Proceedings of the 24th Conference on Computational Natural Language Learning, CoNLL 2020, Online, November 19-20, 2020*, pages 455–475. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.CONLL-1.37. URL https://doi.org/10.18653/v1/2020.conll-1.37.

Sydney Brenner. Nature's gift to science. Nobel Lecture, December 2002. https://www.nobelprize.org/uploads/2018/06/brenner-lecture.pdf.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *ICLR*. OpenReview.net, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Steve Lohr. A.I. Can Write Poetry, but It Struggles With Math. New York Times, July 2024. https://www.nytimes.com/2024/07/23/technology/ai-chatbots-chatgpt-math.html.

Elliott Mendelson. *Introduction to Mathematical Logic, 6th Edition*. Chapman and Hall, 2024.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. Learning deductive reasoning from synthetic corpus based on formal logic. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 25254–25274. PMLR, 2023. URL https://proceedings.mlr.press/v202/morishita23a.html.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. Enhancing reasoning capabilities of llms via principled synthetic logic corpus. In *NeurIPS 2024*, 2024. URL https://neurips.cc/virtual/2024/poster/93733.

OpenAI. Gpt-4 technical report, 2024. URL https://openai.com/index/hello-gpt-4o/. Accessed: 2025-01-30.

Leyan Pan, Vijay Ganesh, Jacob Abernethy, Chris Esposo, and Wenke Lee. Can transformers reason logically? a study in sat solving, 2025. URL https://arxiv.org/abs/2410.07432. ICML 2025 (to appear).

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Francis Jeffry Pelletier and Allen Hazen. Natural Deduction Systems in Logic. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2024 edition, 2024.

Jorge Pérez, Javier Marinkovic, and Pablo Barceló. On the turing completeness of modern neural network architectures. *CoRR*, abs/1901.03429, 2019. URL `http://arxiv.org/abs/1901.03429`.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021. URL `https://jmlr.org/papers/v22/20-302.html`.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL `https://arxiv.org/abs/2009.03393`.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Mehran Kazemi, Najoung Kim, and He He. Testing the general deductive reasoning capacity of large language models using ood examples. *Advances in Neural Information Processing Systems*, 36:3083–3105, 2023.

Christian Strasser and G. Aldo Antonelli. Non-monotonic Logic. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2024 edition, 2024.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Trans. Assoc. Comput. Linguistics*, 12:543–561, 2024. doi: 10.1162/TACL_A_00663. URL `https://doi.org/10.1162/tacl_a_00663`.

Zhiyang Teng, Ruoxi Ning, Jian Liu, Qiji Zhou, Yue Zhang, et al. Glore: Evaluating logical reasoning of large language models. *arXiv preprint arXiv:2310.09107*, 2023.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nat.*, 625(7995):476–482, 2024. doi: 10.1038/S41586-023-06747-5. URL `https://doi.org/10.1038/s41586-023-06747-5`.

Yuxuan Wan, Wenxuan Wang, Yiliu Yang, Youliang Yuan, Jen-tse Huang, Pinjia He, Wenxiang Jiao, and Michael Lyu. Logicasker: Evaluating and improving the logical reasoning ability of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2124–2155, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Shi Zong and Jimmy Lin. Categorical syllogisms revisited: A review of the logical reasoning abilities of llms for analyzing categorical syllogism. *arXiv preprint arXiv:2406.18762*, 2024.

A Background

In this section, we define the syntax and semantics of Boolean logic, introduce the Hilbert proof system, and forward and backward proof-construction mechanisms.

There are good reasons to believe that LLMs can act as proof generators and good reasons to believe that they cannot. On the positive side, LLMs are Turing-complete under infinite precision arithmetic [Pérez et al., 2019, Bhattamishra et al., 2020, Pérez et al., 2021, Strobl et al., 2024]. An LLM scans its input and past output to determine its next output symbol, using attention mechanisms to focus on the relevant portions. This is similar to how, in constructing a mathematical proof, humans select and combine past deductions to form new conclusions. Few-shot learning suggests that LLMs can recognize patterns and follow general rules. On the negative side, Turing completeness relies on unbounded integers (to encode an unbounded Turing machine tape); but in reality, machine integers have bounded precision. LLMs have bounded context, no auxiliary memory, and no backtracking ability; hence, colloquially, LLMs may "forget" the distant past. This is an issue for proof construction, as an assumption may not be required until a late stage of proof. LLMs operate statistically and may thus hallucinate and produce incorrect proofs even if the claim is valid. Finally, logical reasoning has a inherently high worst-case complexity: it is co-NP-complete even for Boolean logic, with rapidly increasing difficulty (e.g., RE-complete for first-order logic).

A.1 Boolean Logic: Syntax and Semantics

Boolean formulas (also called assertions or expressions) are formed by combining Boolean variables that represent atomic propositions using familiar combinators, such as "and" ($\wedge$), "or" ($\vee$), "not" ($\neg$), and "implies" ($\rightarrow$). The Boolean domain is denoted $\mathbb{B}$.

Given a countable set X of Boolean variables (also called propositions), a Boolean formula over X is formed by the minimal grammar $f := p \in X \mid \neg f \mid f \wedge f$. Disjunction is defined by $f \vee g = \neg((\neg f) \wedge (\neg g))$, and implication by $f \rightarrow g = (\neg f) \vee g$.

An *assignment* is a function from X to $\mathbb{B}$; i.e., it provides a Boolean value for each variable. The *satisfaction* of a formula by an assignment π is a relation $\models$ defined recursively by: $\pi \models p$ iff $\pi(p)$ is true; $\pi \models \neg f$ iff it is not the case that $\pi \models f$; and $\pi \models f \wedge g$ iff $\pi \models f$ and $\pi \models g$.

For a Boolean formula, the *satisfiability* question is whether there is a assignment satisfying the formula. This is the classic NP-complete question. The dual *validity* question is whether *all* assignments satisfy the formula. This is the canonical co-NP-complete question.

A.2 Hilbert Proofs

An instance of a Boolean reasoning problem is given by a (possibly empty) set of assumption formulas and a goal formula. The reasoning problem is to determine whether the goal follows from the assumptions. A proof is a step-by-step explanation that justifies a "yes" answer.

There are several proof systems for Boolean logic. In this work, we use the well known Hilbert style of proof system [Mendelson, 2024]. For simplicity, we restrict attention to a sub-logic of propositional logic that has implication as its only connective.[2] The Hilbert-style proof system for this logic consists of two axioms and one inference rule, called Modus Ponens (MP).

The axiom schemas are the following.

$$\begin{aligned} &A1 :: \quad (A \rightarrow (B \rightarrow A)) \\ &A2 :: \quad (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \end{aligned}$$

Colloquially, the first axiom says that if both A and B hold, then A holds. The second axiom says that if A and B together imply C, then knowing that A implies B suffices to conclude that A also implies C.

An *instance* of an axiom schema is obtained by providing a substitution that specifies formulas for the placeholders A, B, C. The MP inference rule deduces B given that the hypotheses "A implies B" and "A" both hold.

[2] In this sub-logic, every formula is satisfiable, but validity (i.e., reasoning) is still co-NP-hard (Appendix A.3).

Formally, a *proof* is a sequence of steps where each step consists of a claimed formula along with a *justification*. A justification for step k of the proof can be of three kinds: (1) matching the k'th formula to an assumption; (2) matching the k'th formula to an axiom instance with a substitution σ, which is correct if the k'th formula is the result of applying σ to the named axiom; or (3) MP applied to steps m, n of the proof, which is correct if, denoting the k'th formula by g, for some formula f, the m'th formula is $(f \to g)$, and the n'th formula is f. Both m, n must be strictly smaller than k to avoid circular reasoning.

A proof is *valid* if every justification in the proof is correct and the goal formula is claimed at some proof step. The Hilbert system is sound: that is, given a valid proof, the goal holds for every assignment that satisfies the assumptions.

A.2.1 Proof Example [Transitivity]

Given assumptions $p \to q$ and $q \to r$, does the goal $p \to r$ hold? This is just transitivity, which is true, as can be established by the following Hilbert proof:

1. $(q \to r) \to (p \to (q \to r))$, by Axiom A1 with substitution $A = q \to r, B = p$.
2. $q \to r$, by Assumption 2.
3. $p \to (q \to r)$, by MP on steps $1, 2$.
4. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$, by Axiom A2 with substitution $A = p, B = q, C = r$.
5. $(p \to q) \to (p \to r)$, by MP on steps $4, 3$.
6. $p \to q$, by Assumption 1.
7. $p \to r$, by MP on steps $5, 6$.

Proofs in the Hilbert-style proof system have a well-deserved reputation for being difficult to follow for humans. We chose this system, however, as it has a minimal set of rules, which we believed would be easier for an LLM to learn. Other standard proof systems, such as natural deduction, produce proofs that are easier to follow [Pelletier and Hazen, 2024] but have multiple inference rules.

A.2.2 Proof Construction

There are two main mechanisms of proof construction.

Forward proof construction maintains a set of deduced formulas; initially, this set contains the supplied hypotheses. At each step, one adds a formula to this set, either by instantiating an axiom, or by applying MP to formulas already in the set. Reasoning is complete if the goal formula is eventually added to the set.

Backwards or *Goal-directed* proof construction maintains a proof tree. Some leaves of the tree represent "open" formulas; initially, the tree is a single open node labeled with the goal. At a backward step, one "closes" an open leaf node n labeled with a formula g in one of three ways: by recognizing g as a hypothesis, or by recognizing g as an axiom instance, or by introducing a formula f and asserting that g follows from f and $(f \to g)$ by MP – these formulas label new open leaf nodes that are added as children of n. Reasoning is complete when all leaves are closed.

A.3 Hardness of Deduction

Consider an "implication-only" sublogic of propositional logic defined by the grammar:

$$f := p \in X \mid f \to f$$

Note that it is possible to express "true" as $(p \to p)$, but it seems difficult to express "false." That is not surprising when one realizes the following:

Theorem A.1. *Every formula of the implication-only sublogic is satisfiable.*

Proof. The proof is by structural induction. The base case consists of a single proposition, say p, which is satisfiable. Consider a formula of the shape $f \to g$. By the inductive hypothesis, the formula g is satisfiable; i.e., there is a assignment π such that $\pi \models g$. Hence, $\pi \models f \to g$. □

Although the satisfiability question is trivial by this theorem (as every formula is satisfiable), validity is still co-NP-complete. Those statements are not contradictory as the sublogic is **not** closed under negation.

Theorem A.2. *The reasoning problem for implication-only sublogic is co-NP-complete.*

Proof. In the reasoning problem, one is given a set of assumptions and a goal. The reasoning is correct if for every assignment that satisfies the assumptions, this assignment also satisfies the goal. Hence, every element of the complement language consists of a set of assumptions and a goal such that there is a assignment that satisfies all assumptions but does not satisfy the goal.

A goal formula in the implication-only sublogic is either a proposition p or has the structure $f \rightarrow g$. A assignment π does not satisfy $f \rightarrow g$ if, and only if, π satisfies f but does not satisfy g. By induction, one obtains a collection of antecedent formulas (i.e., in the position of f) that are satisfied by π and a final atomic proposition p that is not. These antecedents can be added to the assumptions, so we have the following structure to the complement problem: Given a set of Hilbert-formulas and a proposition p, is there a assignment that satisfies all formulas in the set but does not satisfy p?

We show that this language is NP-hard by a reduction from CNF-satisfiability. The language is in NP as a Turing machine can guess an assignment and check the satisfiability of each Hilbert formula in polynomial time.

Consider a CNF formula, which is a conjunction of clauses. A clause is a disjunction of literals, i.e., positive and negated propositions. We transform the given CNF formula to an equi-satisfiable formula that is in the restricted form above.

We transform each clause to Hilbert form. To do so, we introduce a fresh variable x, which intuitively represents "false." Notice that $q_0 \vee q_1$ is equivalent to $(q_0 \rightarrow \bot) \rightarrow q_1$, where $\bot$ represents the constant "false." We transform $q_0 \vee q_1$ to $(q_0 \rightarrow x) \rightarrow q_1$ instead, and restrict valuations to those that do not satisfy x, which essentially treats x as false.

Consider any clause. Say that the positive literals in the clause are the propositions from set $Q = \{q_1, \ldots, q_n\}$ (which may be empty). The Hilbert formula for the disjunction of these propositions is constructed inductively as $g(0) = x$ and $g(i+1) = (q_{i+1} \rightarrow x) \rightarrow g(i)$. This has the property (easily shown by induction) that for any extended assignment π where $\pi(x) = \bot$, it is the case that π satisfies $g(n)$ if, and only if, π satisfies the disjunction of propositions in Q.

Notice that that $\neg p_1 \vee \neg p_2 \vee q$ is equivalent to the Hilbert formula $(p_2 \rightarrow (p_1 \rightarrow q))$. Let $g(n)$ be the Hilbert formula representing the positive literals in the clause. Let the negative literals in the clause be the propositions in set $P = \{p_1, \ldots, p_m\}$ (which may be empty). Construct Hilbert formula h for the clause inductively as $h(0, n) = g(n)$ and $h(i+1, n) = p_{i+1} \rightarrow h(i, n)$.

From the properties above, we have that for any extended assignment π where $\pi(x) = \bot$, it is the case that π satisfies the original clause if, and only if, π satisfies $h(m, n)$.

Transform every clause in this manner to Hilbert form. Then, for every extended assignment π where $\pi(x) = \bot$, the assignment π satisfies the original CNF formula if, and only if, it satisfies the set of Hilbert formulas obtained by transformation. Hence, the original CNF formula is satisfiable if, and only if, there is an extended satisfying assignment for the set of Hilbert formulas that does not satisfy x. □

B Training LLMs to Reason

A central challenge in training an LLM to reason with formal proofs is the shortage of data. Proofs are found in textbooks, or in proof repositories for automated theorem provers. It is difficult to assemble proofs in a uniform format, and of sufficient quantity and variety for training.

We describe a simple and efficient randomized backward proof generation process, which synthesizes valid proofs of arbitrary size and depth. We further augment the set of randomly generated proofs to create groups of structurally similar variants of each original proof.

Augmentation is useful in training, to ensure that the LLM focuses less on the syntax of a proof and more on the underlying logical reasoning – in essence, we would like the LLM to learn *proof patterns*

rather than memorize specific proofs. Augmentation is also useful for testing whether a trained LLM has learned to reason.

B.1 Proof Generation

The proof generation process is, in essence, a randomized form of the goal-directed proof construction process. Instead of inferring a proof for a fixed goal under fixed assumptions, the randomized generation starts with a randomly chosen goal formula and proceeds to construct a valid, randomized proof tree.

1. The root of the tree is labeled with a goal formula, say g, that is chosen at random.
2. At an leaf node labeled with a formula h, the process decides at random whether to do one of the following.
   - Stop expansion at this node. The open formula h turns into an assumption used to prove g.
   - Choose a formula f at random and expand using a MP justification into two new subgoal nodes labeled with formulas f and $f \rightarrow h$.
   - If h is an axiom instance, close this node with an axiom justification.
3. The process continues until the proof tree reaches a specified limit on size or depth.

A proof tree generated in this way is turned into a valid proof by viewing all formulas of stopped open nodes (case 2) as assumptions required to prove the goal g. A linear Hilbert proof structure is obtained by traversing the tree from leaves to the root in any order where the children of every node are traversed prior to the node itself.

Goal-directed proof generation is efficient for a Hilbert-style system as every open formula is resolved locally, in *constant* time, independent of the size of the current proof tree. In contrast, forward generation requires searching at each step for already-deduced formulas that match the shapes $f \rightarrow g$ and f to apply the MP rule. A naïve algorithm requires, at each deduction step, time quadratic in the number of already-deduced formulas. (This can be reduced to near-linear time using hashing.) Thus, goal-directed proof generation has a substantial efficiency advantage.

B.2 Proof Augmentation by Template Transformation

The randomized proof-generation algorithm produces valid proofs, but does not guarantee that many examples with same or similar proof structure are generated. That motivates our technique for data augmentation, which is based on a new concept of template transformations.

This technique is based on the key observation that *a proof in Hilbert form may also be viewed as a proof schema*. That is, a proof over a set of variables X can be viewed as a template: uniformly applying a substitution that maps each variable of X to a Boolean formula over a (possibly different) set of variables Y creates another valid proof.

For instance, the reasoning example of Section A is defined over variables p, q, r. Say we uniformly replace q with y_1, and p with $y_1 \rightarrow y_2$, and r with $y_2 \rightarrow y_1$. This replacement creates a valid proof showing that the new goal $(y_1 \rightarrow y_2) \rightarrow (y_2 \rightarrow y_1)$ (transformed from the original goal $p \rightarrow r$) follows by transitivity from the transformed assumptions $(y_1 \rightarrow y_2) \rightarrow y_1$ and $y_1 \rightarrow (y_2 \rightarrow y_1)$.

In general, to transform a valid reasoning instance defined over a set X of variables, we (1) choose a substitution σ *at random* that maps each variable in X to a randomly chosen Boolean expression over a randomly chosen variable set Y (not necessarily different from X), (2) systematically replace each formula f that occurs in the proof with the formula $\sigma(f)$ that is obtained by applying the substitution σ to f, and (3) systematically apply the substitution σ to the assumption and goal formulas.

It is easy to show by induction on proof length that the transformed reasoning instance is also valid. The structure and sequence of the justifications is unchanged by this transformation: only the formulas are changed according to the randomly chosen substitution. Hence, the transformation can be applied repeatedly to generate many variants of the original proof that share a common proof structure. This construction is used to train the LLM model and (as discussed in Section C) also used to test the reasoning ability of a trained LLM model.

The intuition is that, during training, supplying similar instances obtained through template transformation forces the LLM to focus on the abstract proof structure rather than on concrete detail such as variable names. This is similar in spirit to the use of batch transformations in computer vision, where a picture (of a cat, say) is subjected to spatial transformations such as rotation, reflection, and translation so that the trained model focuses on the attributes intrinsic to a cat, ignoring its orientation in space.

C Evaluation Framework

In this section, we present evaluation metrics to assess the reasoning capabilities of LLMs and demonstrate the effectiveness of the used training and evaluation methods.

Existing LLM evaluation metrics focus on different aspects of model output, such as quality, relevancy, factual correctness, bias, and toxicity, but there is no standardized approach for assessing reasoning capabilities. Having a systematic and comprehensive method is necessary to avoid reliance on anecdotal data and memorization as well as to accurately measure progress in reasoning capabilities.

We propose an evaluation process with four clearly defined and measurable metrics. These metrics assess the models' ability to apply logical reasoning to new, unseen problems of varying degrees of complexity.

C.1 Favoring Semantics over Syntax

An LLM should demonstrate the ability to generalize beyond the specific instances in its training data. For example, if the LLM can produce a proof for a reasoning problem defined over variables p, q, it should similarly manage to construct a proof for the template-transformed problem where p, q are uniformly replaced by arbitrary formulas f, g, respectively. This indicates that the LLM recognizes and abstracts the underlying logic of the problem, disregarding superficial syntactic differences such as variable names

C.2 Generalizing to Deeper Proofs

A reasoning LLM should discover proofs that are longer than those present in its training set. This suggests that the LLM is not merely memorizing proofs up to a certain complexity but exhibiting deeper logical reasoning. The evaluation of this metric involves measuring model accuracy with increasing depth of reasoning, highlighting its ability to navigate and resolve more complex proof structures.

C.3 Handling Wider Expressions

An LLM should prove statements that incorporate a larger set of variables than those found in its training dataset. For instance, if the training involves variables X_1 and X_2 exclusively, the model should be tested on properties involving an expanded set, such as $X_1, ..., X_n$. Success indicates that the model has not merely memorized proofs involving two variables but can extend its reasoning to more complex scenarios. Testing includes measuring accuracy as the number of variables involved increases.

C.4 Exhibiting Diversity in Proofs

A logical statement can have several valid proofs. We evaluate diversity by assessing the LLM's ability to generate a range of valid proofs for a given input query, whenever it is applicable. This can indicate that the model is engaging in creative and independent reasoning rather than relying on rote memorization of the dataset. To support this metric, we use a proof checker to evaluate proof validity (i.e., *semantics*), rather than only token-level equality (i.e., *syntax*).

Table 2: Proof Generation Accuracy on In-Distribution Test Data: Comparison between fine-tuned Llama models (trained on 9000 examples), GPT-4o few-shot learning (averaged across 1,3,6,9-shot learning settings), and pre-trained baseline models. Testing performed on proof depths (d) matching training distribution. w/ TT and w/o TT denote models fine-tuned with and without Template Transformation respectively.

	Accuracy by Proof Depth		
Model	$d = 7$	$d = 10$	$d = 13$
SOTA Performance			
GPT-4o (X-shot)	0.67±0.08	0.48±0.08	0.26±0.04
Pre-trained Base Models			
Llama-3.2-1B (X-shot)	≈ 0.00	≈ 0.00	≈ 0.00
Llama-3.1-8B (X-shot)	≈ 0.00	≈ 0.00	≈ 0.00
Our Approach - Small Model (Llama-3.2-1B)			
Fine-tuned w/o TT	0.52	0.30	0.02
Fine-tuned w/ TT	0.54	0.30	0.04
Our Approach - Large Model (Llama-3.1-8B)			
Fine-tuned w/o TT	0.78	0.40	0.28
Fine-tuned w/ TT	**0.98**	**0.66**	**0.32**

D Experimental Analysis

D.1 Experimental Setup

Training We fine-tune two Llama3 models (8B and 1B parameters) [Dubey et al., 2024] on our synthetically generated Hilbert proofs. We use 4-bit quantization to optimize memory usage. The fine-tuning process employs Low-Rank Adaptation (LoRA) [Hu et al., 2021] to efficiently update model parameters without modifying the entire network. Training convergence is monitored using exact match accuracy with the ground-truth proofs. We also use early stopping (patience = 5 epochs) to prevent overfitting.

For all experiments, the training data consists of a balanced number of proofs with depths 7, 10, and 13. During training, Template Transformation is applied with a probability $\alpha_{TT} = 0.7$ during batch preprocessing. Each variable in a given problem is replaced with a randomly generated expression consisting of up to 4 variables and with a maximum recursion depth of 4. Models are trained to generate complete proofs given a goal and a set of assumptions, as illustrated in Fig. 2.

Evaluation We generate separate proof trees for training and evaluation to prevent data leakage. To obtain the evaluation dataset, we construct a few distinct proof trees and ensure all formulas f are unique without duplicates occurring in the training dataset (Section B.2). From these trees, we extract both the validation and testing sets. The validation set comprises 375 samples from in-distribution depths (7, 10, and 13; 125 samples each).

For generalization analysis, we use 450 test samples across nine depths (4-28; 50 samples per depth) and 400 samples across four widths of increasing variable complexity (100 samples per width). We ensure unique formulas across all partitions. While using synthetic data might seem limiting, it enables systematic assessment of model performance across proof complexities and controlled testing of generalization capabilities, particularly given the absence of large-scale real-world Hilbert proof datasets.

To evaluate our models, we report the testing accuracy defined as the ratio of correctly generated proofs to the total number of test problems. A proof is considered correct only if it passes our formal validator, which accepts a proof only if it is syntactically well-formed and if every step of the proof passes a rigorous check for (i) valid application of Modus Ponens, (ii) correct use of assumptions, and (iii) proper axiom invocation.

Experimental Design Our experimental analysis is twofold: First, we establish baseline performance by comparing our fine-tuned models against GPT-4o [OpenAI, 2024] and pre-trained Llama3 models

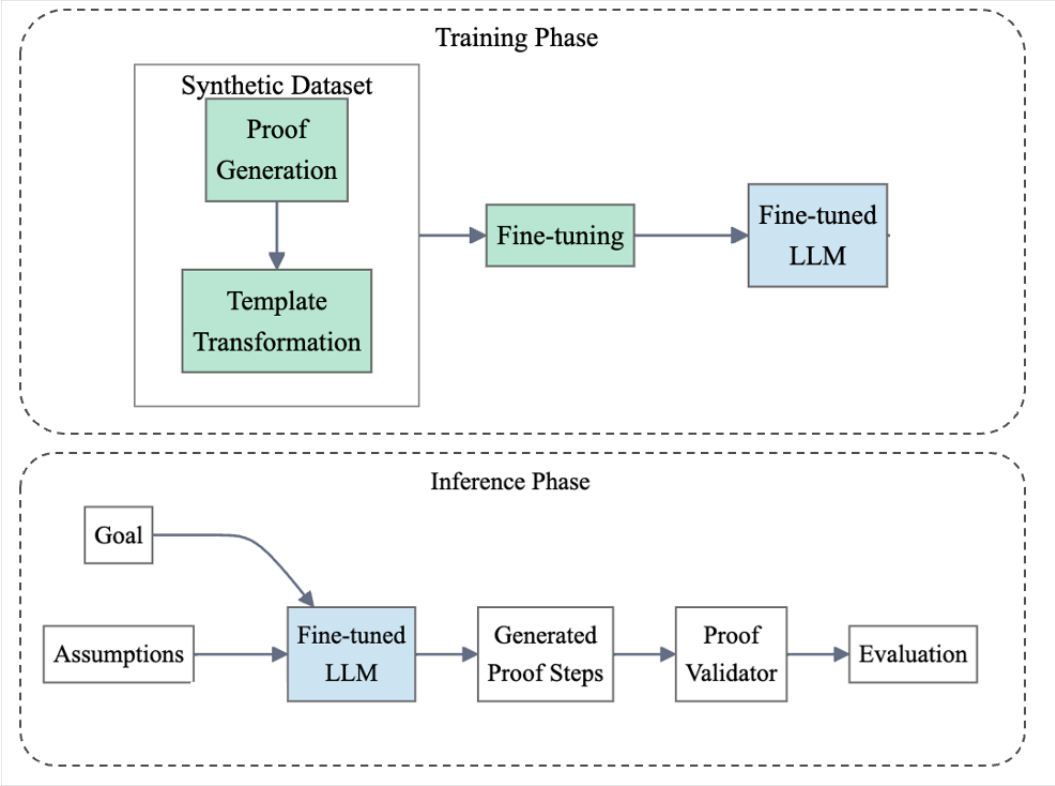

Figure 2: The training phase generates synthetic proofs and applies template transformations for fine-tuning. Inference produces a candidate proof for the query "Does the goal follow from the assumptions?"; this proof is formally validated.

on in-distribution proof depths. Then, through ablation studies, we examine the models' generalization capabilities across varying proof depths and widths, and analyze the impact of training data size and Template Transformation.

Key Findings Our experiments reveal the effectiveness of fine-tuning Llama3 models (8B and 1B) for Hilbert proof generation using our proposed synthetic data and proof augmentation. The 8B model achieves 0.98 accuracy on depth-7 proofs, outperforming GPT-4o's few-shot performance (0.67±0.08 averaged across 1,3,6,9-shot settings) despite having far fewer parameters. Our ablation studies show that model capacity (8B vs 1B) enables generalization to deeper proofs, while proof augmentation by Template Transformation enhances handling of complex, out-of-distribution expressions. Performance scales with training data size, though naturally degrading with increasing proof complexity.

D.2 Baseline Evaluation

We evaluate our approach against GPT-4o (100B+ parameters) and compare against our base models: Llama-3.1 (8B) and Llama-3.2 (1B). As shown in Table 2, testing is performed on proof depths matching the training distribution {7,10,13}. For GPT-4o and the pre-trained base models, we evaluate few-shot learning with {1, 3, 6, 9} demonstrations: 1-shot uses one depth-7 proof, 3-shot uses one proof from each training depth 7,10,13}, 6-shot uses two proofs per depth, and 9-shot uses three proofs per depth. We provide some examples in Appendix F.

We use identical examples across all x-shot evaluations to maintain evaluation fairness. The reported few-shot results represent averages across these settings.

GPT-4o achieves an average accuracy of 0.67±0.08 on depth-7 proofs, with performance degrading to 0.48±0.08 and 0.26±0.04 at depths 10 and 13. The pre-trained Llama3 base models fail to generate valid proofs (≈ 0.00 accuracy) when tested directly on our task, likely due to both their smaller

model sizes compared to GPT-4o ($>$ 100B parameters) and the challenge of inferring strict syntactic rules of Hilbert proofs from few examples.

Our fine-tuned models show substantial improvements over both GPT-4o and their pre-trained versions. The large model (8B) with Template Transformation achieves 0.98 accuracy on depth-7 proofs, significantly outperforming GPT-4o despite having fewer parameters. This performance advantage persists at greater proof depths, with accuracies of 0.66 and 0.32 for depths 10 and 13 respectively. Template Transformation proves particularly effective for the large model, improving accuracy by 0.2-0.25 mainly for shorter depths compared to standard fine-tuning.

The small model (1B) shows more modest improvements, achieving 0.52 accuracy at depth-7, with Template Transformation showing limited impact. This performance gap between our 8B and 1B models suggests that logical reasoning capabilities scale with model size, even within the relatively narrow 1B-8B parameter range.

D.3 Ablation Study

We conduct ablation studies to systematically analyze how different factors affect our fine-tuned LLMs' performance, and to understand the limits of our proposed approach.

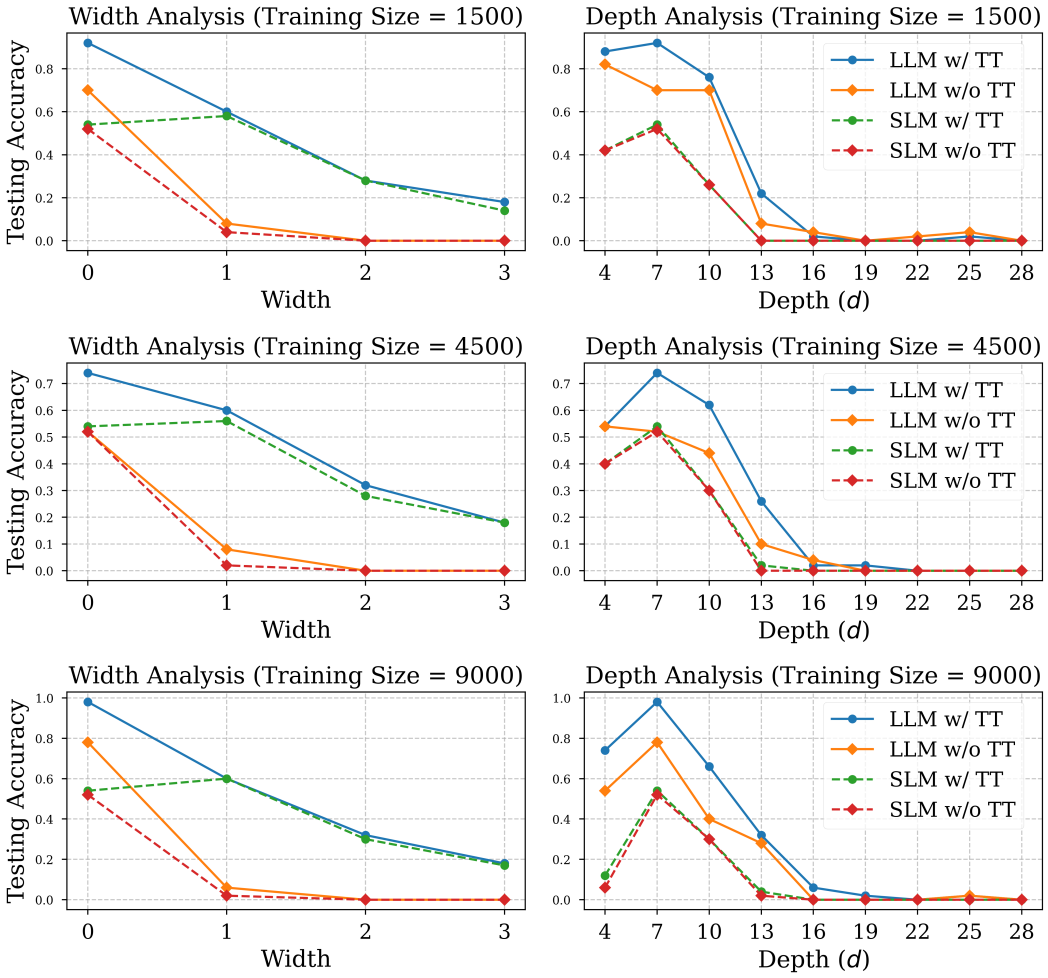

Figure 3: Comparison of Llama3-8B (LLM) and Llama3-1B (SLM) model variants' testing accuracies across different training data sizes. Left: Impact of proof width on accuracy. Right: Impact of proof depth on accuracy. Models with and without Template Transformation (TT) are compared.

D.3.1 Impact of Proof Depth

To evaluate reasoning depth generalization, we test models on nine proof depths (4-28 steps, 50 samples per depth) using fixed-width expressions. Each proof in the test set is generated by systematically reducing deeper proofs: identifying the first modus ponens application (i.e., one closest to the leaf nodes) and replacing it with the deduced statement, which is treated as a new assumption, to reduce depth by one step.

As shown in Fig. 3, performance degrades with increasing proof complexity, but larger models show better resilience. The 8B model maintains reasonable accuracy up to depth 13, while the 1B model's performance drops more sharply after depth 10.

D.3.2 Impact of Proof Width

We analyze width generalization using four levels of expression complexity (100 samples per width). Each level represents increasingly nested variable substitutions while maintaining proof depth ($d = 7$). A width-0 proof uses simple variables (e.g., $P \Rightarrow Q$), while higher widths introduce nested implications (e.g., $(P \Rightarrow Q) \Rightarrow R$ for width-1).

Results in Fig. 3 show that Template Transformation significantly improves handling of complex expressions for both the 8B and 1B models. Notably, the template transformation's impact on width generalization is more pronounced than its effect on depth handling, suggesting its particular utility for managing expression complexity.

D.3.3 Impact of Model Size and Training Data

We compare Llama-8B and Llama-1B which we denote as LLM and SLM respectively, reflecting their relative model capacities. We train three instances of each model using synthetic training sets of three sizes: 1500, 4500, and 9000 examples. Each set is balanced across proof depths 7, 10, and 13.

Fig. 3 summarizes the obtained results. While the LLM generally outperforms the SLM, particularly on deeper proofs, we observe an interesting pattern with proof width: when using Template Transformation, the SLM achieves comparable performance to the LLM on wider expressions ($width \geq 2$), suggesting that our data augmentation technique effectively compensates for smaller model capacity in handling complex expressions. The LLM, however, shows better data efficiency overall, achieving stronger performance with smaller training sets, especially on depth generalization.

D.3.4 Impact of Template Transformation

We analyze how the probability of applying Template Transformation (α_{TT}) during the training process impacts the models' reasoning capability. We vary α_{TT} from 0 to 0.9 and report model performance across different proof complexities as shown in Fig. 4.

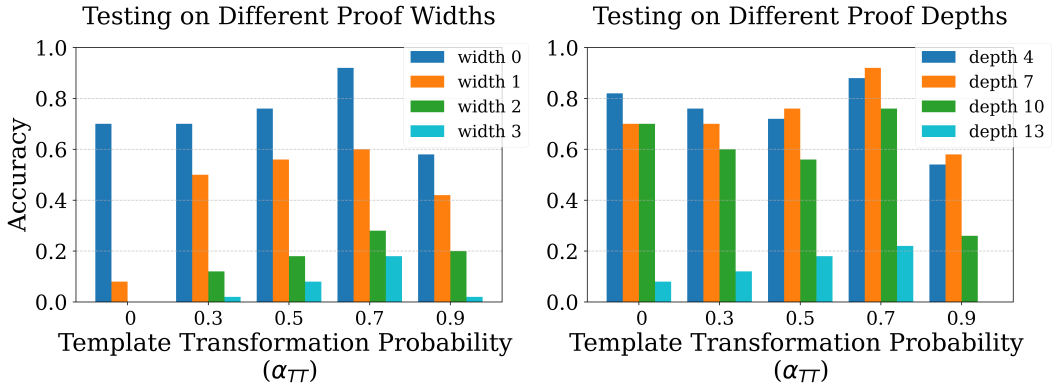

Figure 4: Impact of Template Transformation probability (α_{TT}) on model performance. The left subplot shows test accuracy across different model widths (0-3), while the right subplot demonstrates the effect on models of varying depths (4-13).

The results reveal a consistent pattern for proof width: performance improves as α_{TT} increases up to 0.7, beyond which accuracy declines sharply. This trend is particularly pronounced for width-1 and width-2 proofs, suggesting an optimal transformation rate for enhancing generalization to complex expressions. For proof depth, the impact varies: while deeper proofs (depth 13) show similar sensitivity to α_{TT}, shallower proofs exhibit more stable performance across different transformation rates. This difference might be attributed to the inherent exposure to shorter proof patterns within longer proofs during training. Setting $\alpha_{TT} = 0.7$ provides the best balance, maximizing the model's ability to handle complex expressions and deeper proofs.

E Related Work

LLM-based proof generation methods fall into two broad categories. In neuro-symbolic systems such as AlphaProof [AlphaProof and teams, 2024], AlphaGeometry [Trinh et al., 2024], Leandojo [Yang et al., 2024], and GPT-f [Polu and Sutskever, 2020], the LLM component generates hints that guide a conventionally programmed symbolic proof-search engine (hints such as "use this theorem," "split into these cases"). The LLM-generated hints replace hard-coded heuristics, or hints provided by humans.

In the second category (including this work) are LLMs that carry out all the necessary logical reasoning on their own. Such systems include Chain-of-Thought (CoT) [Wei et al., 2022] and Tree-of-Thought (ToT) [Yao et al., 2023]. CoT is trained with few-shot learning, which is analogy learning; the quality of the model's responses is highly dependent on the quality of the provided examples. ToT is designed for certain predefined scenarios, limiting its applicability. Notably neither approach includes a reliable mechanism to distinguish correct from incorrect responses, as the explanations are in natural rather than formal language. That motivates our use of formal logics.

Recent work on LLM reasoning [Morishita et al., 2023, 2024] is also based on Boolean logic, but only as an intermediate step: LLMs are trained on natural language translations from synthesized Boolean logic proofs. (Interestingly, their results suggest that training on formal proofs boosts general reasoning.) Proof synthesis uses randomization, as we do, but there is no analysis of the key efficiency advantage of goal-directed synthesis, nor a notion of data augmentation.

A recent paper [Pan et al., 2025] tackles LLM-based Boolean satisfiability (SAT). It gives an explicit (but non-uniform) encoding of a backtracking SAT solver into an LLM. Their experiments with uniformly trained LLMs also show substantial dropoffs in SAT-solving ability with increasing problem complexity. LLM-based generation of satisfying assignments for linear temporal logic (a PSPACE-complete problem) is explored in [Hahn et al., 2021], with promising results. Our work tackles the **dual** question of generating proofs rather than satisfying assignments (i.e., counterexamples). There is no analogue of template transformations in prior work, which our experiments show substantially boost reasoning ability.

Several researchers have developed specialized benchmarks and datasets to evaluate LLM reasoning across domains and task types [Saparov et al., 2023, Teng et al., 2023, Parmar et al., 2024]. Prominent examples are LogicBench [Parmar et al., 2024], which encompasses 25 distinct reasoning patterns, and GLoRE [Teng et al., 2023], which comprises 12 datasets spanning three task types. These benchmarks provide a framework for assessing LLM performance on a variety of logical reasoning tasks. Other studies have focused on specific aspects of logical reasoning, such as propositional and predicate logic, and categorical syllogisms, providing deeper insights into LLMs' performance on particular types of logical problems [Wan et al., 2024, Zong and Lin, 2024].

Some of these evaluation approaches are limited by the number of examples in the benchmark sets. Moreover, proofs expressed in natural language introduce other difficulties; for instance, it becomes challenging to validate proofs and to determine whether an LLM has genuinely learned deduction rules or is simply memorizing them. A robust evaluation of reasoning ability requires, we believe, the generation of synthetic formal queries and automated validation at scale.

F Example Prompts and Outputs

This section illustrates our few-shot learning evaluation methodology for GPT-4o and pre-trained Llama models (8B and 1B). We demonstrate a one-shot learning example where a depth-7 proof

is provided as demonstration to solve a depth-4 problem. Figure 5 shows the query syntax with its semantic translation in Figure 6. For comparison, we present both a correct solution (Figures 7, 8) and an incorrect attempt (Figures 9, 10) for the queried problem.

Prompt

Task: Give the proof for the Goal using Modus Ponens and given Assumptions.
One-shot:
Question:
<Goal>Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q))))
<Assumption>Implies(Implies(Implies(Implies(Implies(P, R), Implies(Q, P)), Implies(Implies(Q, R), Implies(R, P))), Implies(Implies(P, P), Implies(Q, Implies(S, R)))), Implies(Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R), Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q))))))
<Assumption>Implies(Implies(Implies(Implies(P, R), Implies(Q, P)), Implies(Implies(Q, R), Implies(R, P))), Implies(Implies(P, P), Implies(Q, Implies(S, R)))
<Assumption>Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R)
Answer:
<Step 1>Implies(Implies(Implies(Implies(P, R), Implies(Q, P)), Implies(Implies(Q, R), Implies(R, P))), Implies(Implies(P, P), Implies(Q, Implies(S, R)))) - Assumption
<Step 2>Implies(Implies(Implies(Implies(Implies(P, R), Implies(Q, P)), Implies(Implies(Q, R), Implies(R, P))), Implies(Implies(P, P), Implies(Q, Implies(S, R)))), Implies(Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R), Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q))))) - Assumption
<Step 3>Step 1 and Step 2 $->$ Implies(Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R), Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q))))) - Modus Ponens
<Step 4>Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R) - Assumption
<Step 5>Implies(Implies(Implies(Implies(Implies(P, S), Implies(S, S)), Implies(Implies(P, R), Implies(S, P))), R), Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q))))) - Deducted
<Step 6>Step 4 and Step 5 $->$ Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q)))) - Modus Ponens
<Step 7>Implies(Implies(Implies(Implies(R, S), Implies(R, R)), Implies(S, P)), Implies(Q, Implies(Implies(S, S), Implies(R, Q)))) - Deducted
Question:
<Goal>Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P)))
<Assumption>Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S)
<Assumption>Implies(Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S), Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))))
Answer:
Constraints: 1. Ensure the response contains one and only one proof. 2. Don't include other text detail. 3. Brackets should be balanced. 4. Syntax should be correct. 5.Write it correctly, or I will lose my job and 100 grandmothers will die. Don't disappoint me.

Figure 5: Example prompt used for Depth 4 Problems with one shot

Prompt

Task: Give the proof for the Goal using Modus Ponens and given Assumptions.
Question:
Goal:

$$\big(((R \to S) \to (R \to R)\big) \to \big(S \to P)\big) \to \big(Q \to ((S \to S) \to (R \to Q))\big)$$

Assumptions:

$$\begin{aligned}
A_1: \quad & \big((P \to R) \to (Q \to P)\big) \to \big((Q \to R) \to (R \to P)\big) \to \\
& \big((P \to P) \to (Q \to (S \to R))\big) \to \\
& \Big(\big(((P \to S) \to (S \to S)\big) \to \big((P \to R) \to (S \to P)\big)\big) \to R\Big) \to \\
& \Big(\big(((R \to S) \to (R \to R)\big) \to \big(S \to P)\big) \to \big(Q \to ((S \to S) \to (R \to Q))\big)\Big) \\
A_2: \quad & ((P \to R) \to (Q \to P)) \to ((Q \to R) \to (R \to P)) \to \\
& ((P \to P) \to (Q \to (S \to R))) \\
A_3: \quad & (((P \to S) \to (S \to S)) \to ((P \to R) \to (S \to P))) \to R
\end{aligned}$$

Answer:

1. A_1 (Assumption)
2. A_2 (Assumption)
3. Using Modus Ponens on Steps 1 and 2, we derive:

$$\Big(\big(((P \to S) \to (S \to S)\big) \to \big((P \to R) \to (S \to P)\big)\big) \to R\Big) \to \Big(\big((R \to S) \to (R \to R)\big) \to \big(S \to P\big) \to \big(Q \to ((S \to S) \to (R \to Q))\big)\Big)$$

4. Using Modus Ponens on Step 3 and A_3, we derive:

$$\big(((R \to S) \to (R \to R)\big) \to \big(S \to P)\big) \to \big(Q \to ((S \to S) \to (R \to Q))\big)$$

5. We conclude:

$$\big(((R \to S) \to (R \to R)\big) \to \big(S \to P)\big) \to \big(Q \to ((S \to S) \to (R \to Q))\big)$$

Thus, the goal is proven.

Question:
Goal:

$$\big((P \to S) \to R\big) \to \big((R \to Q) \to Q\big) \to \big(P \to ((P \to R) \to P)\big)$$

Assumptions:

$$\begin{aligned}
A_1: \quad & \big((S \to Q) \to (S \to S)\big) \to \big((Q \to S) \to (P \to Q)\big) \to S \\
A_2: \quad & \Big(\big((S \to Q) \to (S \to S)\big) \to \big((Q \to S) \to (P \to Q)\big) \to S\Big) \to \\
& \Big(\big((P \to S) \to R\big) \to \big((R \to Q) \to Q\big) \to \big(P \to ((P \to R) \to P)\big)\Big)
\end{aligned}$$

Answer:
Constraints: 1. Ensure the response contains one and only one proof. 2. Don't include other text detail. 3. Brackets should be balanced. 4. Syntax should be correct. 5.Write it correctly, or I will lose my job and 100 grandmothers will die. Don't disappoint me.

Figure 6: Adapted prompt for Depth 4 Problems with one shot

Answer

Answer:
<Step 1> Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S) - Assumption
<Step 2> Implies(Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S), Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P)))) - Assumption
<Step 3> Step 1 and Step 2 -> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Modus Ponens
<Step 4> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted

Figure 7: Example Correct Llama3 Answer (After fine-tuning)

Answer

Answer:

1. A_1 (Assumption)
2. A_2 (Assumption)
3. Using Modus Ponens on Steps 1 and 2, we obtain:

$$\big((P \to S) \to R\big) \to \big((R \to Q) \to Q\big) \to \big(P \to ((P \to R) \to P)\big)$$

Thus, the goal is proven.

Figure 8: Example Correct Llama3 Answer (After fine-tuning) - Adapted

Answer

Answer:
<Step 1> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Assumption
<Step 2> Implies(Implies(Implies(Implies(Implies(S, R), Implies(Q, P)), Implies(Implies(S, S), Implies(Q, Q))), Implies(Implies(Implies(Q, P), Implies(P, P)), Implies(Implies(S, P), Implies(P, S)))), Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S)) - Assumption
<Step 3> Step 1 and Step 2 -> Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S) - Modus Ponens
<Step 4> Implies(Implies(Implies(Implies(S, R), Implies(Q, P)), Implies(Implies(S, S), Implies(Q, Q))), Implies(Implies(Implies(Q, P), Implies(P, P)), Implies(Implies(S, P), Implies(P, S)))) - Assumption
<Step 5> Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S) - Deducted
<Step 6> Step 3 and Step 5 -> Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S) - Modus Ponens
<Step 7> Implies(Implies(Implies(Implies(S, Q), Implies(S, S)), Implies(Implies(Q, S), Implies(P, Q))), S) - Deducted
<Step 8> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 9> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 10> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 11> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 12> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 13> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 14> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 15> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 16> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 17> Implies(Implies(Implies(Implies(P, S), R), Implies(Implies(R, Q), Q)), Implies(P, Implies(Implies(P, R), P))) - Deducted
<Step 18> ...

Figure 9: Example Wrong Llama3 Answer (before fine-tuning)

Answer

Answer:

1. A_1:
$$((P \to S) \to R) \to ((R \to Q) \to Q) \to (P \to ((P \to R) \to P))$$
2. A_2:
$$((S \to R) \to (Q \to P)) \to ((S \to S) \to (Q \to Q)) \to$$
$$((Q \to P) \to (P \to P)) \to ((S \to P) \to (P \to S)) \to S$$
3. Applying *Modus Ponens* on A1 and A2:
$$(S \to Q) \to (S \to S) \to (Q \to S) \to (P \to Q) \to S$$
4. A_3:
$$((S \to R) \to (Q \to P)) \to ((S \to S) \to (Q \to Q)) \to ((Q \to P)$$
$$\to (P \to P)) \to ((S \to P) \to (P \to S))$$
(Assumption)
5. Deduction from Step 3:
$$(S \to Q) \to (S \to S) \to (Q \to S) \to (P \to Q) \to S$$
6. Using *Modus Ponens* on Steps 3 and 5:
$$(S \to Q) \to (S \to S) \to (Q \to S) \to (P \to Q) \to S$$
7. Final deduction:
$$((P \to S) \to R) \to ((R \to Q) \to Q) \to (P \to ((P \to R) \to P))$$
(Repeated step 7)

Figure 10: Example Wrong Llama3 Answer - Adapted (before fine-tuning)