Proof Flow: Preliminary Study on Generative Flow Network Language Model Tuning for Formal Reasoning

Matthew Ho
UC San Diego
La Jolla, CA
mah048@ucsd.edu

Vincent Zhu
UC Santa Barbara
Santa Barbara, CA
vincentzhu@ucsb.edu

Xiaoyin Chen
Mila, Université de Montréal
Montréal, Québec, Canada
xiaoyin.chen@mila.quebec

Moksh Jain
Mila, Université de Montréal
Montréal, Québec, Canada
moksh.jain@mila.quebec

Nikolay Malkin
University of Edinburgh
Edinburgh, UK
nmalkin@ed.ac.uk

Edwin Zhang
OpenAI
San Francisco, CA
edwin@openai.com

Abstract

Reasoning is a fundamental substrate for solving novel and complex problems. Deliberate efforts in learning and developing frameworks around System 2 reasoning have made great strides, yet problems of sufficient complexity remain largely out of reach for open models. To address this gap, we examine the potential of Generative Flow Networks [GFlowNets; Bengio et al., 2021, Hu et al., 2024] as a fine-tuning method for LLMs to unlock advanced reasoning capabilities. In this paper, we present a proof of concept in the domain of formal reasoning, specifically in the Neural Theorem Proving (NTP) setting, where proofs specified in a formal language such as Lean can be deterministically and objectively verified. Unlike classical reward-maximization reinforcement learning, which frequently over-exploits high-reward actions and fails to effectively explore the state space, GFlowNets have emerged as a promising approach for sampling compositional objects, improving generalization, and enabling models to maintain diverse hypotheses. Our early results demonstrate GFlowNet fine-tuning's potential for enhancing model performance in a search setting, which is especially relevant given the paradigm shift towards inference time compute scaling and "thinking slowly." Code: `https://github.com/matt-seb-ho/gfn_ntp`

1 Introduction

Large language models (LLMs) have demonstrated impressive capabilities in pattern recognition and surface-level tasks, but still exhibit significant limitations in reasoning, particularly in complex logical inference and problem-solving tasks. Extending LLMs beyond mere memorization requires enhancing their reasoning abilities through approaches like System 2's "slow thinking," which scales inference time computation to match problem complexity. A promising technique in this space is GFlowNet fine-tuning [Hu et al., 2024], which unlocks new possibilities for search (a fundamental approach to reasoning [Simon, 1983]) by teaching the model to sample trajectories proportional to reward. In this sense, fine-tuning model M with the GFlowNet objective moves the inference time cost of sampling more suggestions ("slow thinking") from M to training time, thus amortizing the cost of inference.

Reasoning benchmarks like GSM8K [Cobbe et al., 2021] and MATH [Hendrycks et al., 2021] are increasingly subject to overfitting as models are trained specifically on these formats and similar math problems, raising concerns about their ability to capture the true generalization ability of models for real-world problem-solving. To address this, we turn to formal mathematics and neural theorem proving (NTP), leveraging proof assistants like Lean [Yang et al., 2023]. Built on dependent type theory, Lean can automatically apply common proof patterns through tactics—commands such as `intro, apply, simp`—which directly manipulate the proof state, transforming theorem proving into a formalized search problem. Unlike standard benchmarks, NTP offers interactive feedback and

38th Conference on Neural Information Processing Systems 2024, Sys-2 Reasoning Workshop.

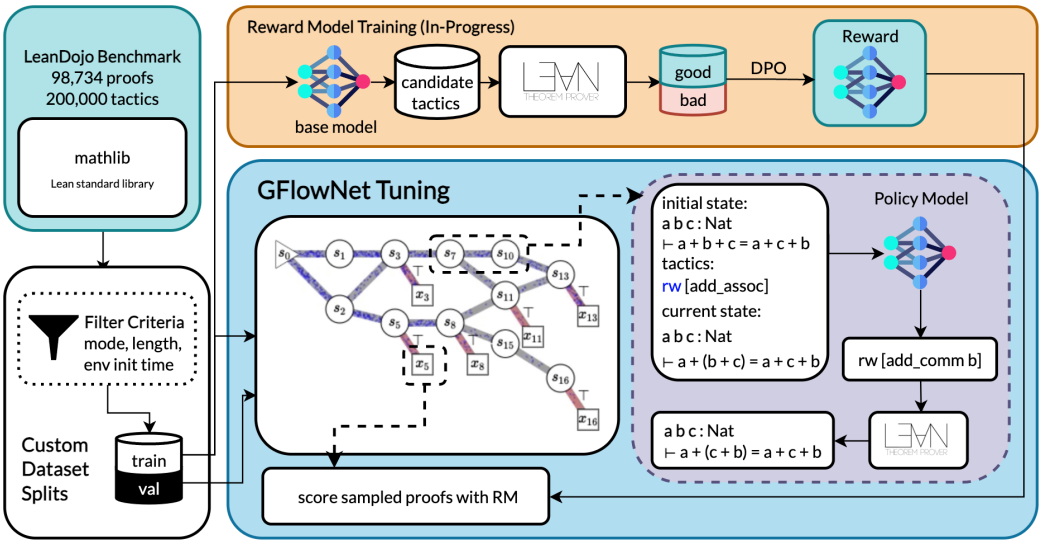

Figure 1: **Proof Flow System**. We extract and filter ground-truth proofs and theorems from LEAN 4's standard math library: mathlib. For reward model training data, we sample candidate tactics using a base model and label with Lean. We use Best First Search [Pearl, 1984] for evaluation. The GFlowNet diagram is a frame of an animation from Bengio et al. [2022]

guarantees correctness, ensuring that flawed reasoning cannot yield correct results—a common issue in iterative self-teaching approaches [Zelikman et al., 2022].

In this paper, we present early in-progress evidence that GFlowNet fine-tuning has potential to accelerate search in challenging reasoning domains like theorem proving. **Our contributions include:**

(1) A study on GFlowNet fine-tuning's promise for accelerating search in a domain challenging even for human experts. (2) An extensible code base integrating GFlowNet with the Lean environment.(3) An ablation study of key interventions in GFlowNet fine-tuning, including reward models and trajectory replay. (4) Early empirical results showing GFlowNet fine-tuning improves exploration and reasoning in neural theorem proving tasks.

2 Methodology

In the context of language modeling, GFlowNets are a maximum entropy RL algorithm for training policies to sample compositional objects with probability proportional to a reward [Bengio et al., 2021, Hu et al., 2024] (see preliminary details in Appendix C). Following past NTP work (Appendix B), and leveraging the interactive features of Lean, we parameterize proof search as a tactic generation problem. GFlowNets model the construction of the object as a terminating trajectory through a graph where edges (actions) specify adding a component to the object and nodes (states) are intermediate states of the object. In this view, NTP can be viewed as composing proofs from component tactics. We refer the reader to Hu et al. [2024] for a detailed discussion about GFlowNets in the context of language model fine-tuning.

We define our forward policy as $P_f(t|s)$ where t is a tactic and s is current proof state, all prior tactics taken, and initial proof state (in Lean, tactic states include the goal). Since GFlowNet graphs allow nodes to have multiple parents (analogously, multiple Lean tactic sequences can lead to the same proof state), optimizing GFlowNet objectives require learning a backwards policy (specifying how reward arrived in current node from). Not only would this be difficult as it requires learning the inverse of the function that the base model learned in fine-tuning, it would also add another source of training instability. To avoid this problem, we use a state encoding that includes the trajectory's history. Instead of representing a partially constructed proof by just its current proof state, we include its initial state and all previous tactics. This enforces a tree structure, making the backward policy trivial by ensuring that each proof state has only one parent.

As for the reward, one option is to use the feedback provided by the Lean verifier as a binary reward. While strictly correct and therefore un-hackable, this binary reward may be too sparse and penalize promising partial trajectories that were unable to complete due to limited search budget. To address

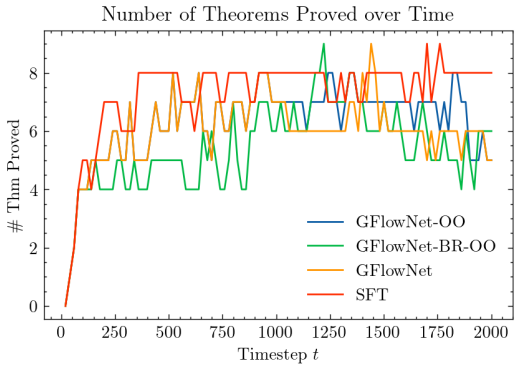

Figure 2: Evaluation on hold-out set of 20 theorems unseen during train time. Validation run every 20 gradient steps. GFlowNet-OO refers to Online Only, GFlownet-BR-OO refers to Binary Reward and Online Only, while GFlowNet refers to the full method. SFT refers to Supervised Fine-Tuning, or just maximizing log likelihood on the ground truth trajectory. For exact theorem names and lengths see Table 1.

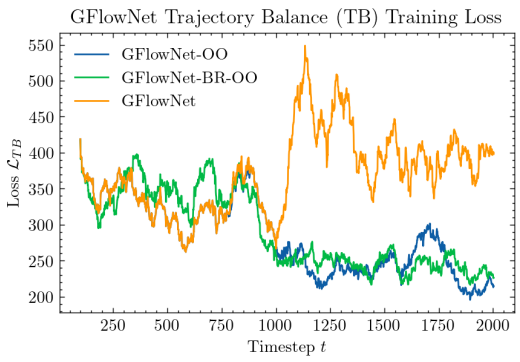

Figure 3: Training Trajectory Balance loss over 2000 gradient steps. Loss is smoothed using simple moving average over 100 steps. Interestingly, GFlowNet observes instability in training around episode 1000. Note that the full GFlowNet is the only off-policy method, as opposed to the on-policy GFlowNet-OO and GFlowNet-BR-OO.

this, we introduce partial reward through a reward model (RM). Here, we take advantage of ReProver's [Yang et al., 2023] training objective (maximizing $P(t|\text{current_proof_state})$ for ground truth proof trajectories) and use it to score the individual tactics in a partial trajectory (note that for the RM, we use the history-less state encoding, or just the current proof state). The trajectory balance [TB; Malkin et al., 2022] learning objective for GFlowNets also requires estimating the log partition function $\log Z$, where Z is the sum of rewards for all terminal states for the given theorem. To that end, for each theorem, we run a forward pass of the policy model on the initial state, and feed the model's final hidden states to a linear layer that predicts $\log Z$. This linear layer is learned simultaneously with the policy.

The trajectories used for training are sampled from the current policy in conjunction with the Lean environment to verify syntactic correctness and yield next states. In additon, we sample ground truth correct trajectories which we are given access to during train time, whereas during the test time evaluation and validation loops, we use standard best first proof search. Please see Appendix D for more methodological details and detailed algorithm pseudocode.

3 Experiments

Data. Our experiments are based on the LeanDojo benchmark Yang et al. [2023]. Starting from the Lean4 `random` splits version, we apply several rounds of filtering to form our train and validation splits (see subsection D.1 for details). The final train and validation splits contain 1K and 20 instances, respectively. **Model.** We conduct experiments initializing our model with ReProver, Yang et al. [2023] a pretrained tactic generator, which was in turn initialized from ByT5-small [Xue et al., 2022] a 350M parameter sequence-to-sequence model whose byte level vocabulary makes it well suited for the unicode-symbol-heavy Lean language. We train using a single A100 40GB GPU. Please see all other hyperparameters in Appendix A. **Ablations.** We conduct ablations over the reward function and replay buffer. In standard setting (GFlowNet)—closest to the original GFlowNet LLM fine-tuning [Hu et al., 2024] setup—each training step samples from the replay buffer with probability 0.5. In the online-only (GFlowNet-OO) setting, the replay buffer is unused– every training step receives reward from a newly sampled trajectory. In the binary-reward and online-only (GFlowNet-BR-OO) ablation, we use a binary reward that gives perfect score to correct trajectories and a length-penalized score to all other outcomes. In all GFlowNet runs, we inject the ground truth trajectory as way to help stabilize training, This particular setting makes GFlowNet most comparable to SFT, which we also include as a baseline. Ongoing experiments explore removing this intervention and scaling train time exploration. **Compute Budget.** Proof search evaluation used much tighter constraints both due to time/resource limits, but also to test the model's efficiency in search. Compared to ReProver's evaluation setting with search budget of 10 minutes and branching factor of 64, we use 30 seconds and branching factor

8. For the fairest comparison, the base model was evaluated using ReProver's original state encoding instead of the history-augmented encoding used for SFT and GFlowNet training runs.

Preliminary Results Analysis. The main results are displayed in Table 1. We observe that under low resource constraints, GFlowNet fine-tuning is capable of enhancing proof search performance, as evidenced with the substantially improved solve rate compared to the base model. However, while promising, we note that the baseline method of Supervised Fine-Tuning is able to match or exceed the GFlowNet fine-tuning in solve rate, pointing towards the necessity for larger compute and further research. The GFlowNet ablations yield largely similar results to the full method. However, both ablations GFlowNet-OO and BR-OO slightly under-perform the full method up to timestep 1000. Interestingly, there is nearly no difference in the loss or performance between OO and BR-OO before step 1000, which is likely due the fact that the RM is only used when a tactic is syntactically correct, which the model fails to generate before step 1000.

While GFlowNet and GFlowNet-BR-OO settings achieve a higher peaks at 9 theorems solved, they occur roughly two-thirds of the way into their runs before regressing through the end of training. One possible explanation is that through the binary reward punishing incomplete trajectories, it discourages exploration and therefore harms test set generalization in more training steps. In addition, the full off-policy GFlowNet experiences training instability, which motivaties future research into instability mitigation. Some potential avenues may include distribution correction [Kumar et al., 2020] or improved replay buffer sampling strategies [Oh et al., 2021]. Indeed, prior works in GFlowNet fine-tuning have shown that replay buffer usage is essential to good runs in practice and our experiment at least confirms that training with replay was substantially faster, completing 2000 steps in .86 the time other runs took. Thus while replaying off-policy proofs introduces instability in the short run, the training speedup is likely worth the tradeoff given strategies to stabilize the off-policyness.

For our low compute budget training and evaluation, SFT yielded better results than GFlowNet. SFT has time and again been demonstrated to be an extremely efficient learning algorithm, and our setting is no exception. We hypothesize that GFlowNet fine-tuning's potential for improving exploration is not fully realized under our current constraints due to insufficiently well-performing prior and reward models, motivating further work in improving both components. Sampling diverse positive samples from the prior is critical for leveraging the GFlowNet objective to sample more diverse samples than a mode-seeking objective such as Policy Gradient [Sutton et al., 1999], which our existing model is unlikely to do under our compute constraints. Thus, scaling the batch size and search budget during both train and inference time will likely lead to more pronounced gains and improvements over the SFT baseline.

4 In-Progress and Future Work, Limitations, and Conclusion

In-Progress and Future Work. Further variations of the existing setting are currently being tested. In addition to continuing to search certain key hyperparameters (e.g. batch size, number of training steps, replay frequency, reward scaling, etc.), there are more involved interventions that are also currently in progress related to the reward model. As GFlowNet fine-tuning explicitly aims to sample in proportion to reward, our method is gated by the quality of our reward model (reward model engineering details in Appendix E). Additionally, other search approaches for exploration during training may be another fruitful direction, such as Monte-Carlo Tree Search (MCTS). As for future work, one important direction is extending GFlowNet fine-tuning across other formal reasoning tasks, moving towards a "universal reasoner" that generalizes across domains. This could involve tasks beyond Neural Theorem Proving (NTP), like probabilistic programming or program synthesis. Furthermore, GFlowNets' ability to generate multiple samples could help quantify uncertainty in reasoning. The GFlowNet framework also holds promise for more complex processes, such as structured chains of thought and long-term planning, offering new possibilities for LLMs in formal and informal environments. In addition, the scaling laws of System 2 reasoning performance with respect to amount of compute budget needs to be analyzed, and how to allocate that budget most efficiently amongst inference search, training search, and base model size.

Limitations and Conclusion. Our experiments, constrained by search budget during training and dataset size, may limit GFlowNet fine-tuning's full potential. Larger datasets and longer training could further amplify its benefits, especially for larger models (stronger priors could find multiple high reward regions in training more easily). Efficient proof state exploration also remains a challenge. We hypothesize GFlowNet objectives for fine-tuning LLMs can serve as a more principled approach for improving exploration in the context reasoning tasks. Our early results on NTP show promise for GFlowNet fine-tuning even under tight train and search budgets. While SFT performs similarly or even better in these low resource regimes, the underlying principles anticipate better results for GFlowNet fine-tuning with scale and we hope to motivate future work towards achieving such a result.

References

E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Y. Bengio, K. Malkin, and M. Jain. The gflownet tutorial, 2022. URL https://milayb.notion.site/The-GFlowNet-Tutorial-95434ef0e2d94c24aab90e69b30be9b3.

S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, T. T. Wang, S. Marks, C.-R. Segerie, M. Carroll, A. Peng, P. Christoffersen, M. Damani, S. Slocum, U. Anwar, A. Siththaranjan, M. Nadeau, E. J. Michaud, J. Pfau, D. Krasheninnikov, X. Chen, L. Langosco, P. Hase, E. Biyik, A. Dragan, D. Krueger, D. Sadigh, and D. Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=bx24KpJ4Eb. Survey Certification.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training Verifiers to Solve Math Word Problems, Nov. 2021. URL http://arxiv.org/abs/2110.14168. arXiv:2110.14168 [cs].

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

A. Hosseini, X. Yuan, N. Malkin, A. Courville, A. Sordoni, and R. Agarwal. V-STar: Training verifiers for self-taught reasoners. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=stmqBSW2dV.

E. Hu, M. Jain, E. Elmoznino, Y. Kaddar, G. Lajoie, Y. Bengio, and N. Malkin. Amortizing intractable inference in large language models. In *International Conference on Learning Representations*, 2024.

A. Kumar, A. Gupta, and S. Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33:18560–18572, 2020.

G. Lample, T. Lacroix, M. anne Lachaux, A. Rodriguez, A. Hayat, T. Lavril, G. Ebner, and X. Martinet. Hypertree proof search for neural theorem proving. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=J4pX8Q8cxHH.

H. Lin, Z. Sun, Y. Yang, and S. Welleck. Lean-STaR: Learning to Interleave Thinking and Proving, July 2024. URL https://arxiv.org/abs/2407.10040v3.

N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022.

Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang. Learning to sample with local and global contexts in experience replay buffer. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=gJYlaqL8i8.

OpenAI. Learning to Reason with LLMs, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley Publishing Company, 1984. ISBN 9780201055948. URL https://books.google.com/books?id=1HpQAAAAMAAJ.

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

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

H. A. Simon. Search and reasoning in problem solving. *Artificial Intelligence*, 21(1):7–29, 1983. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(83)80003-4. URL https://www.sciencedirect.com/science/article/pii/S0004370283800034. Search and Heuristics.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

R. Wang, J. Zhang, Y. Jia, R. Pan, S. Diao, R. Pi, and T. Zhang. TheoremLlama: Transforming General-Purpose LLMs into Lean4 Experts, July 2024. URL http://arxiv.org/abs/2407.03203. arXiv:2407.03203 [cs].

M. Wu, M. Norrish, C. Walder, and A. Dezfouli. Tacticzero: Learning to prove theorems from scratch with deep reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=edmYVRkYZv.

H. Xin, D. Guo, Z. Shao, Z. Ren, Q. Zhu, B. Liu, C. Ruan, W. Li, and X. Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024a. URL https://arxiv.org/abs/2405.14333.

H. Xin, Z. Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, W. Gao, Q. Zhu, D. Yang, Z. Gou, Z. F. Wu, F. Luo, and C. Ruan. DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search, Aug. 2024b. URL http://arxiv.org/abs/2408.08152. arXiv:2408.08152 [cs].

L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*, 2023.

F. Yu, L. Jiang, H. Kang, S. Hao, and L. Qin. Flow of Reasoning: Efficient Training of LLM Policy with Divergent Thinking, June 2024. URL http://arxiv.org/abs/2406.05673. arXiv:2406.05673 [cs].

E. Zelikman, Y. Wu, J. Mu, and N. Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

A Further Experimental Details and Hyperparameters

We conduct full fine-tuning over 2 epochs, or 2K gradient steps with AdamW optimizer, at lr `1e-4` with 0.5 gradient clipping norm, and batches containing 5 sampled trajectories along with the ground truth trajectory. For online trajectories, we temper the generation with probability 0.666 where temperature is uniformly sampled between 0.25 and 1.0. Log reward formulated as shown below, where τ is a trajectory containing states s_i and tactics t_i, $\alpha = 8$, c is max tactic length (88), $l = \frac{1}{n}\sum_{i=1}^{n}\text{len}(t_i)$, and p_{RM} is the reward model.

$$\texttt{LogR}(\tau) = \begin{cases} 0 & \text{if } \tau \text{ completes the proof,} \\ -15 + \alpha \ln \frac{c-l}{c} & \text{if } \tau \text{ ends with LeanError,} \\ \sum_{i=1}^{n} \frac{1}{\text{len}(t_i)} p_{RM}(t_i|s_i) & \text{otherwise} \end{cases}$$

In the binary reward ablation, reward is instead formulated as:

$$\texttt{BinaryLogR}(\tau) = \begin{cases} 0 & \text{if } \tau \text{ completes the proof,} \\ -15 + \alpha \ln \frac{c-l}{c} & \text{otherwise} \end{cases}$$

B Related Works

B.1 Neural Theorem Proving

Several approaches have been explored in the past to enhance the reasoning abilities of LLMs within formal mathematics. Given the high degree of specialization in formal theorem proving and the relative recency of proof assistants, NTP can be considered a low-data domain especially compared to the related but less verifiable wealth of generic coding data. As such, Reinforcement Learning (RL) has been natural approach to work around this constraint as well as leverage the interactivity of proof assistants. Several works have emulated an AlphaZero-like approach of combining MCTS with online training methods such as policy gradients including TacticZero [Wu et al., 2021] and Hypertree Proof Search [Lample et al., 2022]. Most recently, DeepSeek-Prover v1.5 Xin et al. [2024b] trained with GRPO [Shao et al., 2024], an alternative to PPO Schulman et al. [2017] that uses trajectory group rewards to circumvent the need to train a critic model. While promising, these previous works share the long-established objective of RL algorithms: reward maximization. This regime has been shown to encounter issues such as reward hacking and mode collapse [Casper et al., 2023], hindering the learned model's ability to generalize and find novel trajectories as it concentrates probability around the max reward trajectories found in training.

Another major technique employed to combat the low data environment is data augmentation. Synthetic data is deployed in two orthogonal directions. First, works like DeepSeek-Prover v1 [Xin et al., 2024a] introduce new formal math data points by autoformalizing informal (natural language) high school and undergraduate math solutions scraped from the internet. Secondly, works gather further training tokens by augmenting existing theorems and proofs with natural language "thought" annotations. TheoremLlama [Wang et al., 2024], Lean-STaR Lin et al. [2024], and DeepSeek-Prover v1.5 [Xin et al., 2024b] each leverage contemporary base models' improved understanding of natural language and their ability to perform Chain-of-Thought reasoning by injecting natural language explanations between proof steps. These methods have achieved significant improvements, but in each case, the synthetic data is generated from some model, causing the method to be ultimately bottlenecked by the capability of said model.

C Overview of Generative Flow Networks

Generative Flow Networks are a class of probabilistic models designed to sample complex, structured objects through sequential decision-making processes. Unlike traditional generative models or reinforcement learning approaches that aim to maximize expected rewards, GFlowNets aim to sample objects such that the probability of generating a particular object x is proportional to a predefined non-negative reward function $R(x)$. This property makes GFlowNets particularly suited for tasks that require diverse exploration of high-reward regions in the state space.

Flow Conservation and State Flows. The core principle of GFlowNets is the conservation of probability flow through the state space. Each state s has an associated positive flow value $F(s)$, representing the total "probability mass" passing through that state.

Flow Conservation Equation.

For non-terminal states $s \notin \mathcal{S}_F$:

$$F(s) = \sum_{s' \in \text{Ch}(s)} F(s') P_B(s|s')$$

where:

- $\text{Ch}(s)$ is the set of child states reachable from s.
- $P_B(s|s')$ is the backward policy, defining the probability of transitioning from s' to s.

For terminal states $s \in \mathcal{S}_F$:

$$F(s) = R(s)$$

This ensures that the incoming flow equals the outgoing flow at each state, preserving the total probability mass across the network [Bengio et al., 2022].

Detailed Balance Condition. An alternative formulation is the detailed balance condition:

$$F(s) P_F(s'|s) = F(s') P_B(s|s')$$

where $P_F(s'|s)$ is the forward policy, defining the probability of transitioning from s to s'.

This condition ensures consistency in the flow of probability between states in both forward and backward directions.

Flow Conservation and State Flows.

Forward Policy ($P_F(s'|s)$). The forward policy guides the generation of new states (tactics) from the current state:

$$P_F(s'|s) = \frac{F(s') P_B(s|s')}{F(s)}$$

This equation derives from the detailed balance condition and ensures that the forward transitions are consistent with the flow values.

Backward Policy ($P_B(s|s')$). The backward policy is typically defined based on the problem structure and can often be chosen to simplify computations. In the NTP setting, $P_B(s|s')$ could represent the likelihood of reverting a tactic or the inverse of a forward action.

Trajectory Probabilities and Sampling. The probability of a trajectory τ under the forward policy is: $P_F(\tau) = \prod_{t=0}^{n-1} P_F(s_{t+1},|,s_t)$. The induced distribution over terminal states s_n satisfies: $P_F(s_n) = \frac{R(s_n)}{Z}$, where Z is the partition function: $Z = \sum_{s \in \mathcal{S}_F} R(s)$. This means that the probability of sampling a particular proof is proportional to its reward, aligning the sampling process with the objective of exploring high-reward proofs

Learning Objectives. The primary goal is to learn the forward policy $P_F(s'|s;\theta)$, parameterized by θ, such that the induced distribution over terminal states matches the target distribution defined by the reward function.

Trajectory Balance Loss. We employ the Trajectory Balance (TB) loss [Malkin et al., 2022] to train the model:

$$\mathcal{L}_{\text{TB}}(\theta) = \mathbb{E}_{\tau \sim P_F} \left[\left(\log R(s_n) - \log Z + \sum_{t=0}^{n-1} \log P_F(s_{t+1}|s_t;\theta) - \sum_{t=1}^{n} \log P_B(s_{t-1}|s_t) \right)^2 \right]$$

This loss function encourages the model to produce trajectories whose cumulative log-probabilities match the log-reward of the terminal state, adjusted by the partition function. Moreover, TB loss provides better credit assignment along trajectories, leading to more stable and efficient training. This stability is contrasted with traditional reinforcement learning methods, which often suffer from high variance in gradient estimates due to sparse or delayed rewards. The TB loss mitigates this issue by incorporating the reward function directly into the loss, enabling smoother gradients and more consistent updates during training.

D Further Methodology Details

We give our algorithm pseudocode in detail here. `SampleTactic(policy, state)` simply samples a model completion for prompt=state using some temperature.

Algorithm 1 GFlowNet Fine-Tuning Algorithm with TB Loss within NTP Setting

Require: Forward Policy model (policy), Replay Buffer Usage probability p, Theorem Dataset $\mathcal{D}$, `ReplayForward`(τ): gets log probability of old trajectory with current policy, `SampleTrajectory`(theorem, `RM`): samples a proof trajectory and computes $(\log p_f, \log r)$, num trajectories

Initialize `LeanDojo` environment for theorem proving
repeat
  for each theorem $\mathrm{Thm} \in \mathcal{D}$ **do**
    batch $\leftarrow$ []
    if random number $\in [0,1] < p$ **then**
      for trajectory $\in$ [1, 2, ..., num trajectories] **do**
        Select a trajectory $\tau = \{s_0 \rightarrow ... \rightarrow s_n\}$, $\log r$ from replay buffer $\mathcal{B}$
        $\log p_f \leftarrow$ `ReplayForward`(τ)
        batch $\leftarrow$ batch + ($\log p_f$, $\log r$, τ)
      end for
    else
      for trajectory $\in$ [1, 2, ..., num trajectories] **do**
        $(\log p_f, \log r, \tau) \leftarrow$ `SampleTrajectory`(Thm`.initial_state`, policy)
        batch $\leftarrow$ batch + ($\log p_f$, $\log r$, τ)
      end for
    end if
    Add batch to replay buffer $\mathcal{B}$
    Compute Trajectory Balance Loss: $\mathcal{L}_{\text{TB}}$(batch)
    Update model parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{TB}}$
  end for
until $\mathcal{L}_{\text{TB}}$ converged

Algorithm 2 Sample Trajectory Procedure

Require: initial state, policy, max depth, `ComputeLogR`, `SampleTactic`

tactics $\leftarrow$ []
states $\leftarrow$ [initial state]
state $\leftarrow$ (initial state, tactics, initial state)
logprob $\leftarrow$ 0
for step $\in [1,2,...,$`max_depth`$]$ **do**
  tactic, $\log p_f \leftarrow$ `SampleTactic`(policy, state)
  logprob $\leftarrow$ logprob $+ \log p_f$
  next state $\leftarrow$ Lean run tactic on state
  states $\leftarrow$ states + [next state]
  tactics $\leftarrow$ tactics + [tactic]
  state $\leftarrow$ (initial state, tactics, next state)
end for
return logprob, `ComputeLogR`(states, tactics), tactics

D.1 Dataset Filtering

The filtering criteria includes the following: (1) Proof Style: Lean's tactic mode can be interwoven with standard term mode. For our experiments we only consider proofs that are entirely driven by tactics. (2) Proof Length: for practical and generalization purposes, we examine limited trajectories and only consider theorems where the ground truth proof requires three or fewer steps (3) State and Tactic length: memory constraints limited our study to theorems where the ground truth proof has state and tactic lengths under a threshold (under the ReProver tokenizer: 900 state tokens, 90 tactic tokens) (4) Dojo Initialization Time: LeanDojo wraps a Lean REPL with a thorough dependency management system. We consider only theorems whose environment can be initialized within 5 seconds. Due to time and resource constraints, our early experiments are run on a small subset of `mathlib` theorems.

Length	Theorems	Base	SFT	GFN-BR-OO	GFN-OO	GFN
1	Finset.card_insert_of_not_mem					
	Part.map_map	■	■	■	■	■
	Ideal.isCompactElement_top					
	CategoryTheory.Iso.trans_conjAut	■	■	■	■	■
	List.Nodup.erase_get					
	Nat.dist_eq_sub_of_le		■	■	■	■
2	div_nonneg		■	■	■	■
	SimpleGraph.commonNeighbors_top_eq	■	■	■	■	■
	zero_le_four					
	CategoryTheory.exact_kernel					
	Matroid.Restriction.finite					
	Even.sub_odd		■	■	■	■
	Batteries.RBNode.Ordered.setRed	■	■	■	■	■
3	List.lookupAll_length_le_one					
	Real.hasDerivAt_negMulLog					
	Real.Angle.expMapCircle_add					
	Int.ediv_two_mul_two_add_one_of_odd		■	■		■
	Real.log_of_pos		■	■	■	■
	IsCompactlyGenerated.Boolean...					
	Complex.natCast_cpow_natCast_mul					
Total		4/20	9/20	9/20	8/20	9/20

Table 1: **Max Validation Results.** We ran proof search evaluation every 20 training steps. This table records the theorems proved in run's best solve rate validation step. Green cells indicates a proof was found under the search budget described in section 3

Our train set includes 1000 theorems with an extremely small validation split containing 20 theorems. Both contain a uniform distribution of proof lengths.

D.2 Addressing Mode Collapse with GFlowNet-Based Sampling

One of the central challenges of reinforcement learning in proof search is the occurrence of mode collapse [Casper et al., 2023], where the model prematurely converges on a small set of tactics and fails to adequately explore alternative solutions. This significantly hampers the model's ability to generalize, as it becomes locked into a narrow trajectory of proof strategies. By using GFlowNets, we can ensure that the model continues to explore diverse proof strategies by assigning probabilities to entire proof sequences based on their compositional reward.

In contrast to temperature sampling, where randomness is injected at the token level, GFlowNets allow for sampling from the actual sequence distribution [Yu et al., 2024], grounded in the likelihood of success in the proof domain. This approach both mitigates the risk of mode collapse and establishes a more structured investigation schema. The probability assigned to each tactic is directly informed by the reward associated with that tactic's effectiveness in driving the proof forward, making the sampling process more principled and aligned with the task's objectives.

D.3 Amortizing LLM Inference for Efficient Sampling

System 2 reasoning systems generally rely on increased inference time compute. This typically manifests in using more tokens or searching more candidates. The improved exploration efficiency of GFlowNets can also be understood as "amortized LLM inference" [Hu et al., 2024]. Specifically, there are certain intractable posteriors such as $P(Y|X)^{1/T}$ (typically approximated using token-wise tempering), or in our case sampling proofs proportional to their reward. We can use Monte Carlo methods to approximate sampling this, but through training with the GFlowNet objective, we effectively replace the compute spent on inference time Monte Carlo sampling to GFlowNet fine-tuning's train time, which consequently gets amortized over every inference.

D.4 Scalability of Inference Time Computation

The power of scaling inference time compute has been recently been demonstrated with o1 [OpenAI, 2024]. An additional benefit of the GFlowNet approach is its compatibility with this new paradigm. By amortizing inference, we effectively reduce the computational cost of sampling additional proof

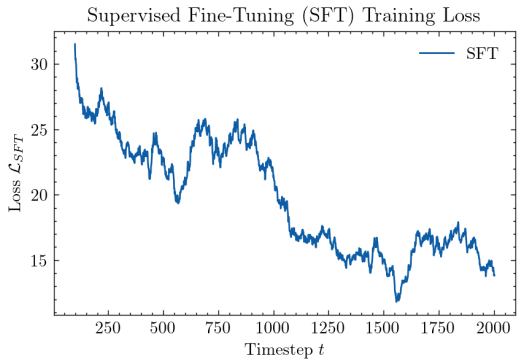

Figure 4: SFT Baseline Train Loss over Time

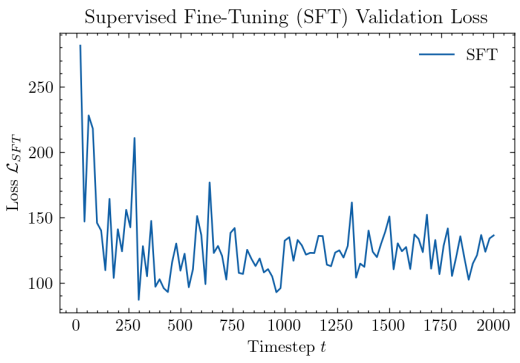

Figure 5: SFT Baseline Validation Loss over Time

strategies, making it feasible to explore a larger portion of the proof space within practical time limits [Hu et al., 2024].

D.5 Supervised Fine-Tuning (SFT) Training Loss

The training loss curve for the Supervised Fine-Tuning (SFT) baseline illustrates steady convergence over the course of training, as seen in Figure 4. The model starts with relatively high loss values, reflecting the initial difficulty in predicting the correct proof tactics. However, as training progresses, the loss steadily decreases, indicating that the model is learning to generate more accurate tactic sequences.

The loss function used for SFT is the standard cross-entropy loss, which measures the difference between the model's predicted probability distribution for each tactic and the actual distribution from the ground truth. A major advantage of SFT is its stability and efficiency, as it does not suffer from the variance typically associated with reinforcement learning-based approaches, such as GFlowNet fine-tuning or PPO.

The validation loss, shown in Figure 5, follows a similar trend to the training loss, with gradual improvement over time. This indicates that the model is not overfitting to the training data, but rather learning generalizable patterns. The gap between training and validation loss remains small throughout the process, which supports the observation that SFT, while not necessarily improving exploration, remains a highly effective method for stabilizing learning.

E In-Progress Work

E.1 Reward Model Engineering

In our empirical analysis, we employed a base reward model (RM) as the verifier. However, in line with recent advancements in informal mathematical verifier training [Hosseini et al., 2024], we are working towards integrating more robust reward models. We have established a comprehensive pipeline for reward model training, consisting of data collection, data filtering, proof sampling, verification, training algorithms, and reward model evaluation. For training, we initially adopted a Supervised Fine-Tuning (SFT) approach, a well-established method for training language models. Additionally, we explored Direct Preference Optimization (DPO) [Rafailov et al., 2023], a technique that has gained significant attention in recent years.

While we could trivially use randomly shuffled tactics as negative examples for DPO, this would represent "easy" negatives where simple pattern matching could likely yield a decent performance. To avoid learning shallow but ultimately unhelpful patterns, we prioritize gathering "hard" negatives. By using the base reward model to sample proof trajectories and collecting tactics from failed proofs (as verified by Lean) we can identify incorrect tactics that the model assigns high likelihood– constituting a somewhat on-policy approach. One challenge in this data collection process is in annotating tactics. Lean directly provides a trajectory-level annotation, but there is ambiguity in what the incorrect tactic could be. To solve this, we need some degree of exploration. To determine if tactic t which yields state s is truly a negative tactic, we can explore s and determine if a proof can be found from there. This is inherently limited by the model's proof search capability as well as the search budget, as falling

short in either category could yield false negatives, but it is useful nonetheless for reducing noise. Conversely, suppose we annotate t as a positive tactic because we found a proof τ starting from s. This could still be a false positive if τ immediately undoes t, returning to state s' (i.e. t ultimately did not contribute to the proof). There are some countermeasures that depend on tracking state depths, but we leave a more complete solution to future work.

Reward Model	Llemma-7B Prompt	DeepSeek-Prover Prompt
SFT	86.6%	87.9%
DPO	87.3%	86.4%

Table 2: **Reward Model Evaluation Comparison.** The table displays the accuracy of reward models in evaluation based on the fine-tuning method (SFT vs DPO) and the formatting for prompting (Llemma vs DeepSeek-Prover). The base model for these experiment is DeepSeek-Prover v1

Early results from SFT and DPO training yield mixed results. We found the relative performance of the methods to be sensitive to adjustments in the prompt format. We attribute the inconsistency to the low amount of exploration we used in annotating the training set and are currently preparing a scaled up version.

E.2 Further Baselines

For the early results, we primarily compare GFlowNet fine-tuning's efficacy to SFT. A perhaps more comparable baseline still remains: Proximal Policy Optimization (PPO) [Schulman et al., 2017]. As one of the most widely used forms of reward-maximizing reinforcement learning, we are working to implement a PPO training for the NTP setting.

Proximal Policy Optimization seeks to optimize an objective that balances the trade-off between exploration and exploitation of the current policy π_{old} to maximize rewards and discover new policies π_θ. This balance is maintained using clipped probability ratios, which prevent updates from straying too far from the current policy. The standard PPO objective is expressed as:

$$L(\theta)=\mathbb{E}_{s_t,a_t\sim\pi_{\text{old}}}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}\hat{A}(s_t,a_t),\,\text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)},1-\epsilon,1+\epsilon\right)\hat{A}(s_t,a_t)\right)\right],$$

where ϵ is a hyperparameter that sets the clipping range, and $\mathbb{E}_{s_t,a_t}$ refers to the expectation over an on-policy batch of samples. The state-value function $V(s)$ estimates the expected cumulative reward an agent can obtain starting from state s, assuming it follows a given policy π, which maps states to actions. The action-value function, or Q-function $Q(s,a)$, estimates the expected cumulative reward starting from state s, taking action a, and then following the policy π. $\hat{A}(s_t,a_t)$ represents the advantage function, which is calculated as $A(s,a)=Q(s,a)-V(s)$.