LOCAL LOOK-AHEAD GUIDANCE VIA VERIFIER-IN-THE-LOOP FOR AUTOMATED THEOREM PROVING

Sara Rajaee[*†]
University of Amsterdam
s.rajaee@uva.nl

Kumar Pratik[*], **Gabriele Cesa, Arash Behboodi**
Qualcomm AI Research[‡], Amsterdam
{kpratik, gcesa, behboodi}@qti.qualcomm.com

ABSTRACT

The most promising recent methods for AI reasoning require applying variants of reinforcement learning (RL) either on rolled out trajectories from the model, even for the step-wise rewards, or large quantities of human annotated trajectory data. The reliance on the rolled-out trajectory renders the compute cost and time prohibitively high. In particular, the correctness of a reasoning trajectory can typically only be judged at its completion, leading to sparse rewards in RL or requiring expensive synthetic data generation in expert iteration-like methods. In this work, we focus on the Automatic Theorem Proving (ATP) task and propose a novel verifier-in-the-loop design, which unlike existing approaches that leverage feedback on the entire reasoning trajectory, employs an automated verifier to give intermediate feedback at each step of the reasoning process. Using Lean as the verifier, we empirically show that the step-by-step local verification produces a global improvement in the model's reasoning accuracy and efficiency.

1 INTRODUCTION

As the new applications of modern machine learning are emerging in various scientific and engineering domains, automated mathematical theorem proving has garnered interests from both machine learning researchers and mathematicians. Many ongoing efforts leverage reinforcement learning and expert iteration, inspired by the success of methods like AlphaZero, to build models that search the proof space and provide step wise or holistic solutions (Lample et al., 2022; Xin et al., 2024a; Gloeckle et al., 2024; Anthony et al., 2017; Silver et al., 2018). These solutions are usually verified by formal proof verification systems like Lean (Moura and Ullrich, 2021) or Coq (Coq Development Team, 2024). Relying on Reinforcement Learning (RL) is advantageous in terms of data efficiency but comes with high computational and training costs (Gloeckle et al., 2024). Part of this complexity is related to the necessity of rolling out the proofs and computing rewards from successful episodes.

In contrast, ReProver (Yang et al., 2023) takes a simpler supervised training approach, specifically imitation learning, paired with premise retrieval methods. The key components of this proof system are as follows: theorem that we would like to prove, tactics that are actions toward the final proof and itself consist of set of goals to be proven, premises that are used to prove goals, and state of the proof which includes the set of goals that are still unproven. The approach consists of retrieving the relevant premises from a database given the final theorem and the state of the proof, and then using ReProver to provide tactics for getting to the next state. The proof terminates when all the goals are proven. The method achieves competitive performance with an order of magnitude smaller complexity and training time (Gloeckle et al., 2024). While the computational cost and simplicity

[*]Equal contribution.
[†]Work done during an internship at Qualcomm AI Research, Amsterdam.
[‡]Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

of ReProver are appealing, we empirically observed that many failure cases of ReProver are due to syntactically incorrect tactics or tactics that are not applicable to the current state of a proof. This has detrimental effects on the beam search performed at inference time, as many beams result in invalid tactics that need to be verified by Lean, thus taking away time from exploring more promising tactics.

To preserve the desirable computational efficiency of ReProver and simultaneously address this problem, the natural choice is to fine-tune the model to remove syntactic errors and increase the number of useful tactics for the proof at each state. Recently, feedback-based alignment has been gaining traction in various other similar fields such as automated code generation and various preference optimization methods where the rewards come either from human feedback (RLHF) or other reward models (Ouyang et al., 2022; Ziegler et al., 2019; Rafailov et al., 2024). Since applying many such reinforcement learning methods for training large models can be complex and expensive, various methods have been introduced in the literature with moderate complexity, among which we can refer to Direct Preference Optimization (DPO) (Rafailov et al., 2024) and Group Reward Preference Optimization (GRPO) (Shao et al., 2024). In context of mathematical reasoning and theorem proving, many works emphasized the importance of trajectory-level preferences in mathematical problem solving with large language models (LLMs) (e.g. see Xiong et al. (2024) and Preference Optimization paragraph in Sec. 2). Even when the complexity is saved in the RL training algorithm, computing these trajectory level preferences can incur additional complexity. This discussion extends to more general episodic reasoning tasks with stepwise verification where the model needs to provide outputs that are both syntactically and semantically correct and useful for solving the problem at hand.

Our contribution. In this work, we aim to address the above issues by fine-tuning a pre-trained model which listens and uses the feedback from the tool, in this case Lean, during the training. At each step, our framework, called LeanListener, obtains feedback on the generated tactics directly via its interaction with the Lean software, and performs policy optimization with a reward that is designed based on Lean feedback. Given the pre-trained ReProver model from Yang et al. (2023), we sample different tactics from its output for each proof state in the training set and use Lean feedback to compute the reward. The reward consists of a negative return for invalid tactics, a positive one for applicable tactics, and a return based on the number of remaining unsolved goals. The RL training is done using GRPO. First note that the sampling step for ReProver's output can yield applicable and new tactics that differ from the provided human label. Therefore, it helps the data efficiency of the method by exploring and adding new tactics like what we see in expert iteration. Second, unlike methods like Process-supervised Reward Model (PRM), we do not compute the step-wise reward based on the full trajectory information and only rely on *local look-ahead* feedback from the number of remaining and unsolved goals in the next immediate steps, and therefore, we address the complexity of the trajectory based preference association. Thanks to this fine-tuning strategy, we expect the model to rank valid and effective tactics higher than invalid ones, even if they were not previously observed in the human-labeled trajectories. As a result, our fine-tuned model can make better use of the beam search used at inference time. Besides, as we will show in our numerical results, the local look ahead, online training with Lean in loop, and GRPO are crucial components in improving the final performance of the model. To summarize, our contributions are as follows. We propose a framework for efficient training of the ReProver to leverage the feedback provided by the external tool, in this case, the Lean solver. We use GRPO for training the model using the reward that is based on the applicable tactics and the number of unsolved goals. The GRPO training is particularly beneficial compared to DPO training in our case. We show that the training based on offline dataset generation of positive-negative samples is ineffective, and online training is crucial. The online verification from the tool automatically adds new tactics to the training data. Finally, we can obtain noticeable gains by computing the reward using local look-ahead without recourse to the trajectory level preference.

2 Related Works

In this section, we discuss the closest works related to our paper. We included an extended version of the related works in App. A, and in particular summarize some works in Table 3.

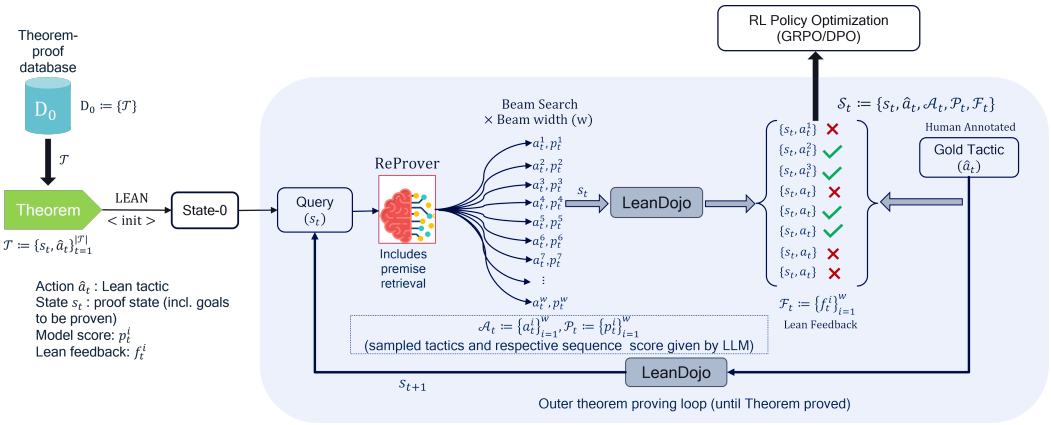


Figure 1: We start with a dataset of human-annotated theorems and their proofs, including the gold tactics, in Lean. During training, the ReProver retrieves the relevant premises from the premise database, and based on those and the state of the proof generates the next tactics. The generated tactics are verified by Lean, and its feedback is used by RL training loop to fine-tune the ReProver. The loop continues using the steps from the human annotated tactics until proof termination.

Theorem Proving. Machine learning is currently used in automatic theorem proving to generate the whole proof, or provide assistance in sequential proofs by finding the right premise or right tactics, or assisting in theorem prover specific formalization of mathematical statements, as well as combination of these ideas; see Xin et al. (2024a;b); Polu and Sutskever (2020); Jiang et al. (2022a); Wang et al. (2023); Jiang et al. (2022b); Wu et al. (2022) for some pointers. We review some of these works in more details. A key component for theorem proving is the formal proof management systems and verifiers like Lean Moura and Ullrich (2021) and Coq Coq Development Team (2024). Lean has received particular attention in the mathematics community for example by its use in verification of some components in the condensed mathematics research program Scholze (2022). We use Lean in this work, and a framework that consists of a premise retrieval part and a prover that generates step-wise proof tactics. In our paper, we are focused on the logic behind tactic generation. In this sense, our approach can be complementary to any work on retrieval and search steps.

Machine learning can be used to find useful premises, definitions and theorems Irving et al. (2016). For Coq framework Coq Development Team (2024), the authors in Blaauwbroek et al. (2024) introduce a graph neural network for embedding the new definitions using a hierarchical representation based on graph representations of theorems and definitions in the Tactician platform Blaauwbroek (2024). This enables them to use recent proofs and theorems in Coq, while kNN is used for the more recent tactics written by the user. In Thor Jiang et al. (2022a), the authors introduce a class of methods to use automated theorem provers for premise selection. LeanDojo Yang et al. (2023) provided a framework that retrieves the premise from a database of Lean premises and uses ReProver to generate tactics for theorem proving in Lean. LeanAgent Kumarappan et al. (2024) adds a dynamic database to LeanDojo, which enables the continual learning of the agent. Besides, a new curriculum learning based on the difficulty of the theorems is also added.

Searching over different proofs and tactics is another component explored in various works, for example Polu et al. (2023); Xin et al. (2024b); Lample et al. (2022). As discussed in Polu et al. (2023), any RL algorithm for theorem proving should address two challenges, namely an infinite dimensional action space, and the absence of a natural opponent for self-play. Therefore, the authors suggest using expert iteration Anthony et al. (2017), which amounts to iteratively fine-tuning a based model and searching to generate correct proofs. They also introduce `lean-gym` to facilitate the search procedure in Lean. HyperTree Proof Searchs (HTPSs) was introduced in Lample et al. (2022) which is a new search algorithms within an online reinforcement learning procedure. The key idea is that the search is represented over a hypergraph, where a policy network generates tactics composed of sub-goals, and then each goal is expanded for proof using new tactics. The provability of each goal is approximated using a critic. The idea of keeping the visit counts, action values and their statistics follows the similar MCTS procedure. In Gloeckle et al. (2024), the authors introduce a reinforcement learning based framework for theorem proving in Lean 4, which consist of using a programming

interface based on Aesop for proof search organization, the HTPS (Lample et al. (2022)) procedure with an online reinforcement learning step. In Zhao et al. (2023), the authors use subgoal learning from reinforcement learning to decompose an informal LLM generated proof into subgoals and verify using a verifier. They also use a diffusion model for demonstration organization.

The transformer-based LLMs are used also for theorem proving. For example GPT-*f* was introduced in Polu and Sutskever (2020) using Metamath as the formalization framework. They showed iterative training a value function on proofs generated by the prover can continually improve the performance. Another example is LLEMMA, which is based on pretrained Llama Code Azerbayev et al. (2024).

DeepSeek-Prover Xin et al. (2024a) generated Lean4 proof data by translating natural language problems into their formal versions in Lean4, and the model produces whole proofs in single turns. DeepSeekV1.5 Xin et al. (2024b) provides a middle ground by generating the whole proof, verifying it via Lean, and then truncating the proof until the first error. The authors propose additional techniques relying on proper appending of previous states and including truncate-and-resume in the Monte-Carlo Tree Search (MCTS) procedure. Additionally, DeepSeekV1.5 leverages verification feedback from Lean on whole-proofs to improve model's *alignment with the formal structure* of Lean. In particular, it uses the Group-Relative Policy Optimization (GRPO) algorithm originally introduced for DeepSeek-Math in Shao et al. (2024), which removes the need for a separate critic model, thereby simplifying the reinforcement learning pipeline and reducing its computational cost. In comparison, we consider a *step-wise* approach, leveraging the verifier feedback at each intermediate step of the proof.

Preference Optimization. Other works in the literature have considered the idea of model alignment to improve the reasoning capabilities of language models, too. In general, there are online version of Reinforcement Learning with Human Feedback (RLHF) as in Bai et al. (2022); Ouyang et al. (2022), and offline versions with either an explicit reward model (e.g. Christiano et al. (2017); Ziegler et al. (2019)) or an implicit reward model (e.g. DPO Rafailov et al. (2024)) to encode the preference. The later method directly optimizes the model without training any independent reward model. In particular DPO uses Bradley-Terry preference model Bradley and Terry (1952) and the fact that the optimal solution to the KL-constrained reward maximization objective is known in closed form to simplify the training loss without dependence on a reward model Rafailov et al. (2024). Many follow-up works explored DPO variations and its shortcomings such as reward over-optimization, for example by introducing Kahneman-Tversky Optimization (KTO) Ethayarajh et al. (2024) and introducing trust-region based versions of previous alignment models Gorbatovski et al. (2024).

In the context of reasoning and, in particular, mathematical problem solving, many recent works have explored different forms of direct preference optimization, for example Xiong et al. (2024); Yuan et al. (2024); Jiao et al. (2024); Cobbe et al. (2021); Lightman et al. (2023); Wang et al. (2024); Pang et al. (2024); Chen et al. (2024); Lu et al. (2024b). The authors in Xiong et al. (2024) introduce a multi-turn version of DPO to use feedbacks from the verifiers, in their case the code interpreters, particularly for multi-turn scenarios, which requires trajectory-level preference optimization. The trajectory-level preference can be obtained by dataset labels of the gold answers or Outcome-supervised Reward Models (ORMs) Cobbe et al. (2021); Lightman et al. (2023). A more fine-grained step-wise supervision can be used as in PRMs Lightman et al. (2023) or by leveraging trajectory level preferences Wang et al. (2024). The idea of ORM and PRM has also been originally discussed in Uesato et al. (2022). These works focus on mathematical reasoning, rather then formal theorem proving, which is the focus of our work. In our work, the theorem prover automatically provides the preference, which is used either during training or for an offline generated dataset. Furthermore, our work follows PRM philosophy as we consider feedback from Lean at each step of proof generation.

3 LEANLISTENER

In Interactive Theorem Proving (ITP), an LLM-based prover interacts with an external proof assistant and receives feedback on the steps to be taken to prove the given theorem. In this work, we use the proof assistant software *Lean*[1] (de Moura et al., 2015; Moura and Ullrich, 2021) as the formal environment which LLMs employ to verify each proof step in a proof sequence. In particular, we used the open-source LeanDojo framework (Yang et al., 2023), which provides toolkits to interact with Lean, as well as readily extracted data and pre-trained models.

[1]We use Lean 4, and for brevity, simply refer to it as Lean in the rest of the manuscript.

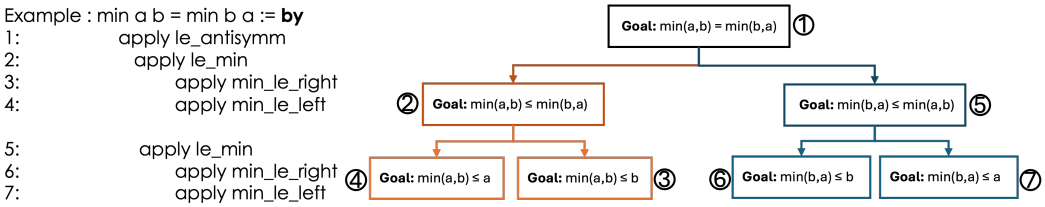


Figure 2: On the left, a proof example in Lean shows the min function is symmetric, and on the right, its proof tree. The numbers next to the boxes refer to corresponding line numbers in the written proof.

We model the theorem proving task as a Markov Decision Process (MDP), a tuple (S, A, P, R), in which S is the set of proof states, A is the set of all possible tactics (i.e., proof steps), P is the transition probability function that describes the likelihood of moving from one state to another given a specific tactic, and R is the reward function that assigns a value to each state transition. An agent interacts with Lean to prove a theorem T by iteratively observing the current "proof state" s_t and performing an action a_t, known as a "tactic". This action transitions the proof to the next state s_{t+1}. Lean evaluates each tactic, either accepting or rejecting it, and provides additional feedback, such as the number of goals remaining to be proven. Our model leverages this feedback during training and for generating subsequent tactics. It is important to note that Lean is a formal language with strict syntax rules, meaning only certain tactics are applicable in each state. At each step, the agent relies solely on Lean's feedback about the current state to generate the next tactic until the theorem T is proven or the model exhausts the predefined time limit. Figure 2 presents a Lean proof example. Finally, the sequence of state-tactic pairs $\{\langle s_t, a_t \rangle\}_t$ composes the proof of the original theorem T.

In general, a certain state s_t can include multiple (sub-) goals, representing a number of independent statements to be simultaneously proved in order to prove the original statement. An intuitive example is given by application of an *induction* tactic, which turns a single statement into two independent ones (the base case and the inductive step). Hence, the application of a tactic a_t on a certain state s_t can prove certain goals, but also turn some into more sub-goals. As a result, the number of sub-goals $\mathcal{G}(s_{t+1})$ can increase, decrease or remain unchanged after the application of each tactic.

Tactic prediction is one of the primary tasks in training LLM provers (Yang et al., 2023; Welleck and Saha, 2023; Lample et al., 2022). In a supervised learning framework, during training, the model is fed the current state (s_t) and asked to predict a tactic (a_t). Recent studies have shown that providing related premises, in addition to the current state, can also enhance the model's performance (Mikuła et al., 2023; Yang et al., 2023). In the inference phase, however, for each theorem, the model employs an inference time compute technique. It observes the current state (s_t) and generates a set of tactics ($\{ a_0, ..., a_k \}$) using *beam search* with a size of k. Building a proof search tree, the tactic generator interacts with the Lean assistant, starting with the tactic with the highest accumulative log probability (best first search). The returned state can be an error state if the tactic execution is unsuccessful, e.g., due to timeout or inapplicable tactic. In this case, the model explores the next tactic in the queue. This process continues until we reach a valid next state (s_{t+1}) or prove the given theorem. It is worth mentioning that the generated proof by the model can be different from the human-written one in the dataset. As many inference-time compute approaches, the model (generator) utilizes feedback from an external expert (verifier) and explores creative proofs at cost of extra computation at inference.

While the pre-training objective is a supervised sequence-to-sequence task, in the evaluation phase, the model acts like an RL agent, which receives feedback on its actions (generated tactics) and uses it to choose the next action via the search strategy explained above. In this work, we resolve this discrepancy and bring the proof-tree expansion based exploration to the training regime, leveraging the feedback provided to each tactic to construct step-wise rewards. Inspired by human preference in LLMs alignment, we present a novel framework, LeanListener, in which we align the pre-trained model with guidance from Lean in a step-wise scenario. In contrast with similar works with RL training like Xin et al. (2024b), we only need a single step look ahead, and do not need the full trajectory to compute the rewards. In what follows, we explain our LeanListener framework in detail.

3.1 METHODOLOGY

Our LeanListener framework is based on employing external proof assistant feedback as guidance signal. To bridge the discrepancy between the seq2seq-based training and beam search-based proof full tree expansion during inference, we bring the per-step proof-tree expansion based exploration to the training regime. Aside from transitioning to the next proof state, applying a tactic in Lean results in valuable information, which can be used to score the efficacy of the tactic at the concerned proof step, and consequently guide the model toward proving theorems. For example, a syntactically valid tactic can be gauged for the number of sub-goals in the theorem it helps resolve.

Lean feedback includes helpful information that can be employed to guide the model toward proving theorems, such as if the generated tactic is applicable to the current state and how many (sub-)goals are solved by applying the tactic. Then, LeanListener utilizes this to reward the model not only for generating an applicable and syntactically correct tactic but also for encouraging the model to generate tactics that solve more (sub)goals. This results in a more efficient proof-generation process.

A straightforward approach to do so is employing Direct Preference Optimization (DPO) (Rafailov et al., 2024), a popular lightweight but effective algorithm used to align LLMs with human preferences, which is often replacing reinforcement learning based solutions with separate reward models like RLHF. The key ingredient in DPO is creating a static preference dataset in the form of pairwise comparisons $D = \{(x_i, y_i^+, y_i^-)\}_i$ such that for each *prompt* x_i, the generated *output* y_i^+ is preferred over y_i^-. DPO directly optimizes the policy π_θ (possibly, initialized to a reference policy π_{ref}) via the following loss based on the static dataset:

$$L_{DPO}(\pi_\theta, \pi_{ref}) = -\mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y^+|x)}{\pi_{ref}(y^+|x)} - \beta \log \frac{\pi_\theta(y^-|x)}{\pi_{ref}(y^-|x)}\right)\right] \quad (1)$$

where σ is the logistic function. The DPO preference dataset can be built based on applicable/not applicable tactics as the positive and negative samples in a pair. In essence, the model can be trained to prefer the applicable tactics, i.e., tactics that lead to a valid proof state, over inapplicable tactics, i.e., tactics that lead to an error state. The unsophisticated DPO pairing configuration will intrinsically reward all the applicable tactics equally, irrespective of whether it takes the proof toward a conclusive state or not. However, a more intricate scoring of tactics can be performed by looking deeper into the Lean feedback, such as the number $\mathcal{G}(s_{t+1})$ of theorem sub-goals that remains to be proven at step s_{t+1}. To craft a more differential scoring of applicable tactics, as reward r_t for a tactic a_t, we choose $R(a_t; s_t) = \textit{softplus}\left(\mathcal{G}(s_t) - \mathcal{G}(s_{t+1})\right)$, where $\textit{softplus}(x, \beta) = \frac{1}{\beta}\ln(1 + e^{\beta x})$. We set β to 0.5. We invariably score the invalid tactics with a score of 0. Such a scoring strategy rewards, and hence, trains the model to prioritize generating tactics that help resolve more sub-goals.

To take the most out of the Lean feedback and incorporate the number of proven (sub-) goals by the tactics as well, we need a more sophisticated objective. In this regard, we propose to use *Group Relative Policy Optimization* (GRPO) (Shao et al., 2024), a variant of reinforcement learning (RL) Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017), in which each tactic is rewarded based on the sub-goals criterion discussed above. For each prompt q (current state s_t), GRPO samples a group of outputs $\{o_1, o_2, \cdots, o_G\}$ (generated tactics) from the old policy $\pi_{\theta\text{old}}$ and maximizes the policy model with the following objective:

$$\begin{aligned} J_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta\text{old}}(O|q)} \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Bigg\{ \\ \min\left[\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta\text{old}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t},\ \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_\theta^{\text{old}}(o_{i,t}|q,o_{i,<t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right] \\ -\beta D_{\text{KL}}[\pi_\theta||\pi_{\text{ref}}]\Bigg\} \end{aligned} \quad (2)$$

Where π_θ and $\pi_{\theta\text{old}}$ are current and old policy models, and q, o are questions and outputs sampled from the dataset and the old policy $\pi_{\theta\text{old}}$, respectively. ϵ and β are hyper-parameters, and $\hat{A}_{i,t}$ is

the advantage calculated based on the reward for each sampled tactic. More specifically, assume the reward value for all generated tactics in a beam search with a size of w is represented by $r = \{r_1, r_2, \cdots, r_w\}$. Then, the advantage $\hat{A}_{i,t}$ is the normalized reward $\hat{A}_{i,t} = \tilde{r_i} = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$. Following Shao et al. (2024), we estimate the KL divergence with the following equation:

$$D_{\text{KL}}(\pi_\theta \,||\, \pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - \log\left(\frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})}\right) - 1$$

We use the pre-trained ReProver model from Yang et al. (2023) both as a base reference model for π_{ref} and to initialize the policy to optimize π_θ.

3.2 Dataset Curation

For both DPO and GRPO finetuning, we build our training dataset using the training split of LeanDojo Benchmark (Yang et al., 2023) to avoid data contamination as the ReProver model has already been pre-trained on it. Inspired by the online-RL training paradigms for eliciting reasoning in LLMs (Gloeckle et al., 2024), we, too, build our training paradigm in an online setting. During online training, we update the model used in the beam search to generate tactic proposals for every 50 training iteration with the target model. We also experiment with the offline setting, where we use the pre-trained ReProver model to generate the training data offline. We discuss the relative advantages of online over offline training in Table 1 and Sec. 4.

To generate the fine-tuning data, we parse the human-annotated proofs in the LeanDojo benchmark using either the reference ReProver model (offline) or the target model (online). More specifically, for each proof state s_t in human-annotated theorem proof in training split, we sample $w = 8$ tactics $\{a_t^j \sim \pi_{ref}(a|s_t)\}_j^w$ via *beam search* and sort them by their likelihood score given by the sampling model. Additionally, we look for the presence of the ground truth tactic, i.e., gold tactic, in the beam search proposals: if the golden tactic $\widehat{a_t}$ for the current state s_t happens to be absent from the top w proposals, we append it as a $w + 1 = 9$-th tactic at the end of the list (i.e. with lowest likelihood); this means states can have either 8 or 9 tactic proposals for training. The next step involves accruing feedback from the Lean by individual application of each proposed tactic on the current proof state.

Strategy	dropout	Prec.@8 ↑	MAP	MRR	Len. Valid Tactics	Len. All Tactics	% 0-precision steps ↓
Offline DPO (zero acc.)	$p = 0.3$	37.75	60.75	64.43	26.66	111.20	29.10
Offline DPO (hard)	$p = 0.3$	35.71	59.90	63.42	28.59	116.00	29.80
Offline DPO (hard)	/	34.86	57.92	61.08	29.37	115.39	32.14
Online DPO (hard)	$p = 0.3$	44.75	64.99	71.22	19.72	27.44	11.88
Online GRPO	$p = 0.25$	**51.01**	**70.09**	**77.05**	**16.26**	**19.87**	**7.44**
ReProver (base model)	/	40.77	59.85	65.70	18.08	22.34	12.74

Table 1: This table reports different metrics to measure the step-wise performance. Prec.@8: percentage of valid tactics among the top 8 tactics sampled at each proof state. MAP: mean average precision percentage. MRR: percentage of mean reciprocal rank. Len. Valid Tactics: average length (in terms of tokens) of the valid generated tactics. Len. All Tactics: average length of all generated tactics. %0-precision steps: fraction of states with no valid tactics.

Dataset for DPO To construct the dataset of *pairwise* comparisons D required by DPO, for each state s_t we need to label its tactics as the positive and negative sample using Lean feedback. A tactic is positive if it is syntactically correct and applicable to the current state s_t by Lean, and it is negative if it gets rejected by Lean. We consider three strategies in the pair creation. **random**: we pair each negative tactic with a positive tactic at random. **zero accuracy**: each negative tactic is paired with the highest-likelihood positive one among those mistakenly ranked lower. **hard**: similar to zero accuracy, but we pick the lowest-likelihood positive tactic. In practice, we also include some

minor heuristics to avoid resampling the same positive tactics too often, see Algorithm 1. Finally, for data augmentation purposes, rather than using the original prompt comprising the state and the retrieved premises $x = (s_t, p_t)$, we dynamically generate a new prompt like in Yang et al. (2023) by applying random dropout on the retrieved premises. Then, once a positive tactic y^+ is chosen, and the augmented prompt x is generated, we add the tuple (x, y^+, y^-) to the dataset D. As a result, each proof state s_t appears in the dataset for each negative tactic the reference model generated for it. In the end, the offline preference dataset consists of 251k triplets (x, y^+, y^-). For online DPO, we keep the same procedure, except for the reference model used for generating the beam-search tactic proposals per proof step which gets updated every 50 iterations with the target model. We describe precisely this dataset generation procedure in Algorithm 1.

Dataset for GRPO For the GRPO scenario, the dataset is being built dynamically throughout the training. More specifically, for each state in the training set, the model generates a set of tactics using beam search. Then, using the reward function described in Sec. 3.1, each generated tactic is scored. The more (sub)goals a tactic solves, the greater its reward. Since the policy model is being updated through GRPO objective, the model's generated tactics change over time. Similar to the online case in DPO, we update the model used for sampling tactics every 50 training steps with the on-policy target model.

3.3 Training Setting

We use the pre-trained generator model in ReProver from Yang et al. (2023) both as a reference model for π_{ref} and to initialize the policy to optimize π_θ. The generator is an encoder-decoder Transformer based on ByT5 (Xue et al., 2022). It is accompanied by a premises retrieval that provides the most related premises as input. More technical details on ReProver can be found in Yang et al. (2023). We fine-tune the pre-trained ReProver model using the described datasets for the 10k steps, AdamW optimizer with a learning rate of $2.25e-6$, and a batch size of 16. The overall fine-tuning in LeanListener framework takes about 40 A100 days. All the experiments have been carried out using HuggingFace TRL trainers.

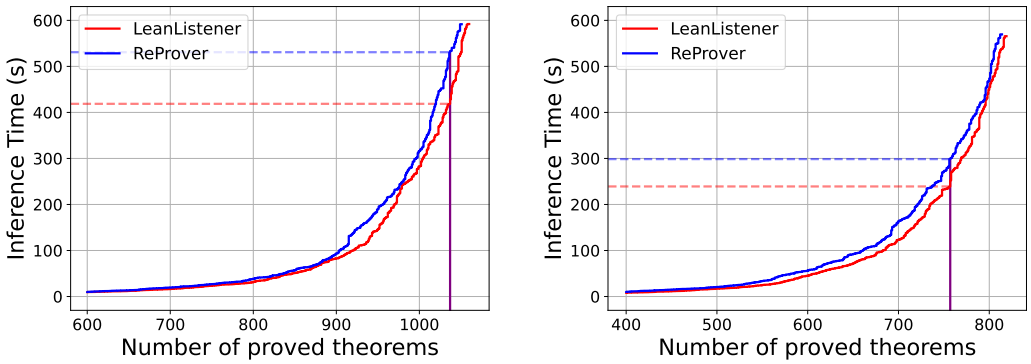
Model	Policy Opt. Method	Pairing Strategy	Online	`random`	`novel premises`
`tidy`				23.8	5.3
GPT-4				29.0	7.4
ReProver (w/o retrieval)				47.6	23.2
ReProver				51.2	26.3
ReProver*				52.76	40.86
LeanListener (Ours)	DPO (binary)	rand.	×	35.99	
		zero acc.	×	33.18	
		hard	×	31.27	
		rand.	∨	50.25	
		zero acc.	∨	50.90	
		hard	∨	50.85	
	GRPO (#sub-goals)	-	∨	**53.21**	**41.11**

Table 2: **Pass@1** (%) performance on the LeanDojo benchmark on the `random` and `novel premises` splits. The performance of the first four baselines is the one reported in Yang et al. (2023), while ReProver* is the newly provided pre-trained model, which we evaluated ourselves.

4 Experiments

We study the effectiveness of LeanListener in different aspects on the test set of LeanDojo benchmark. The test set has two variants, `random`, where the whole dataset is randomly split into training, validation, and test set, and the more challenging `novel premises`, where the test set includes premises that do not appear in the training[2]. In the following experiments, we first verify whether

[2] Yang et al. (2023) only released the model trained on the `random` split, which we use as base for our method. We report the performance of the methods also on the `novel premises` test despite the possible overlap with the training set since this testset is still more challenging than the `random` one. See github issue.



(a) `random` split (b) `novel premises` split

Figure 3: To evaluate the inference speed, we visualize the number of theorems proven by each model within different inference time limits. While the overall number of theorems proved within 10 minutes by the two models is comparable, we see *LeanListener* is consistently faster, which means, it takes significantly less time to solve the same number of theorems (see the purple lines for example).

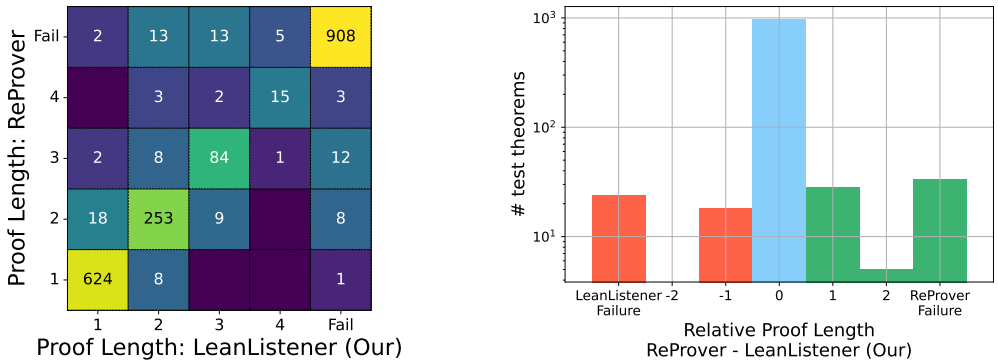
incorporating Lean feedback enhances the quality of generated tactics. Then, we examine the performance on theorem proving. Finally, we study the improved theorem proving speed of our method.

Tactic validity. To begin with, we investigate to what extent adding DPO and GRPO during the training pipeline increases the number of valid tactics generated by the model. A tactic is valid if it is applicable to a state by Lean, even if it does not lead to a proof. The validity is also about the syntactic correctness of the tactic. We expect that if the model generates more valid tactics in the best first search, the chance of proving the given theorem, as well as the efficiency of the proof, will increase.

Table 1 presents the numerical results on the validity of the generated tactics under different scenarios using DPO and GRPO. In this table, we compare the state-wise metrics which we expect to correlate with our training objective. First of all, using the DPO objective, we observe that all offline strategies fail to improve the number of valid tactics generated by the model, while the online strategy significantly improves the number of valid tactics, and reduces the number of states with no valid tactics. Online GRPO provides the best results. It is interesting to observe that the length of valid tactics decrease in the best models, thereby, reducing unnecessarily complex tactic generation.

Performance evaluation. Moving forward, we investigate whether improved tactic generation results in stronger reasoning capabilities or not. Table 2 presents the final theorem proving the performance of our models with the pre-trained ReProver baseline along with GPT-4 (Achiam et al., 2023) and `tidy` (which is a non-machine learning and heuristic-based approach), on the LeanDojo benchmark when using the *random* split. We first observe that while DPO training on binary feedback improves the number of valid tactics, it always leads to a small degradation in the number of proven theorems. We believe that there is spurious correlation between the length of the tactics and their validity on the paired samples in DPO training, which leads to a bias toward generating longer and not necessarily useful tactics. GRPO addresses effectively this issue by online training and avoiding paired samples for training, as well as employing more informative feedback about the number of solved sub-goals. Indeed, we find that this finer reward, which incorporates the number of solved sub-goals, is more effective and leads to an increased number of proven theorems.

Inference time. In the previous experiments, we show that LeanListener generates a more efficient search tree by having more valid tactics. Now, we want to evaluate if this approach leads to a more efficient inference time. Figure 3 compares the inference time in LeanListener and ReProver as the baseline model. As can be observed, LeanListener proves more theorems in up to 20% less time.



(a) Per-theorem proof-length analysis

(b) Per-theorem relative proof-length comparison

Figure 4: For each valid theorem in the test set, we plot and compare the proof-length performance for both the *LeanListener* and *ReProver* models. 4(a) depicts the proof lengths for the proofs generated by the *ReProver* model on the y-axis, and by the *LeanListener* model on the x-axis. The plot shows a higher density above the diagonal, indicating that the proofs generated by *LeanListener* generally have fewer proof steps than those generated by *ReProver* for the same theorem. 4(b) illustrates the relative proof-length advantage of the *LeanListener* method over *ReProver*. For each theorem, we plot the difference in proof lengths generated by both models, observing that instances where *LeanListener* has a positive advantage outnumber those where it does not.

Proof length In addition to the aggregate Pass@1 performance metric discussed earlier, we present an analysis of the proof lengths generated by the LeanListener and ReProver models. Generally, concise proofs, i.e., those with fewer proof steps, are considered more efficient and desirable than lengthier ones. Our aim here is to provide insights into the quality of proofs generated by both models. As shown in Figure 4, our LeanListener consistently produces more concise proofs compared to the base ReProver model.

An interesting observation in Figure 4(a) is the last row, which corresponds to proofs that were proved in just one step by the ReProver model. In this category, LeanListener fails to prove only one theorem, which is intuitive. Since LeanListener is trained with a one-step look-ahead process, it excels at proving theorems that can be solved in a single step. We observe that the number of failed proofs increases for theorems requiring more than one proof step. This suggests potential for further improvements in LeanListener's performance with training that incorporates more than one step look-ahead.

5 Conclusions

In this paper, we propose a novel RL-based framework, *LeanListener*, based on the provided feedback by an external automatic verifier to improve trained LLMs in automatic theorem proving. LeanListener considers the general applicability of tactics as well as their local effectiveness, i.e., their impact on the number of unproven (sub)goals in a proof sequence, to fine-tune theorem provers. Such step-wise reinforcement with the corresponding reward efficiently harnesses the benefits of the trajectory level guidance without outcome-dependent reward computation. Our experimental results show that LeanListener not only surpasses the considered baselines in proving more theorems but also does so faster in less inference time.

REFERENCES

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. Technical report, OpenAI, 2023.

A. Alfarano, F. Charton, and A. Hayat. Global lyapunov functions: a long-standing open problem in mathematics, with symbolic transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 6 Nov. 2024.

T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. M. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4WnqRR915j.

Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, N. Joseph, S. Kadavath, J. Kernion, T. Conerly, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, T. Hume, S. Johnston, S. Kravec, L. Lovitt, N. Nanda, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, B. Mann, and J. Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv [cs.CL]*, 12 Apr. 2022.

L. Blaauwbroek. The tactician's web of large-scale formal knowledge. *arXiv [cs.LO]*, Jan. 2024.

L. Blaauwbroek, M. Olšák, J. Rute, F. I. S. Massolo, J. Piepenbrock, and V. Pestun. Graph2Tac: Online representation learning of formal math concepts. *ICML*, 235:4046–4076, Jan. 2024.

R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324, Dec. 1952.

G. Chen, M. Liao, C. Li, and K. Fan. Step-level value preference optimization for mathematical reasoning. *arXiv preprint arXiv:2406.10858*, 2024.

W. Chen, X. Ma, X. Wang, and W. W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 22 Nov. 2022.

P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Adv. Neural Inf. Process. Syst.*, 30, 2017.

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. *arXiv [cs.LG]*, 27 Oct. 2021.

Coq Development Team. The Coq proof assistant, Dec. 2024.

L. M. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The lean theorem prover (system description). In *CADE*, 2015. URL https://api.semanticscholar.org/CorpusID:232990.

K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela. KTO: Model alignment as prospect theoretic optimization. *arXiv [cs.LG]*, 2 Feb. 2024.

L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 3 July 2023.

F. Gloeckle, J. Limperg, G. Synnaeve, and A. Hayat. ABEL: Sample efficient online reinforcement learning for neural theorem proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=kk3mSjVCUO.

A. Gorbatovski, B. Shaposhnikov, A. Malakhov, N. Surnachev, Y. Aksenov, I. Maksimov, N. Balagansky, and D. Gavrilov. Learn your reference model for real good alignment. *arXiv [cs.LG]*, 15 Apr. 2024.

Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, M. Huang, N. Duan, and W. Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*, 13 Oct. 2023.

X. Guan, L. L. Zhang, Y. Liu, N. Shang, Y. Sun, Y. Zhu, F. Yang, and M. Yang. RStar-math: Small LLMs can master math reasoning with self-evolved deep thinking. *arXiv [cs.CL]*, 8 Jan. 2025.

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

A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzyg'o'zd'z, P. Milo's, Y. Wu, and M. Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. *Neural Inf Process Syst*, abs/2205.10893:8360–8373, May 2022a.

A. Q. Jiang, S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, Sept. 2022b.

F. Jiao, C. Qin, Z. Liu, N. F. Chen, and S. Joty. Learning planning-based reasoning by trajectories collection and process reward synthesizing. *arXiv preprint arXiv:2402.00658*, 2024.

A. Kumarappan, M. Tiwari, P. Song, R. J. George, C. Xiao, and A. Anandkumar. LeanAgent: Lifelong learning for formal theorem proving. *arXiv preprint arXiv:2410.06209*, 2024.

X. Lai, Z. Tian, Y. Chen, S. Yang, X. Peng, and J. Jia. Step-DPO: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*, 2024.

G. Lample, T. Lacroix, M.-A. Lachaux, A. Rodriguez, A. Hayat, T. Lavril, G. Ebner, and X. Martinet. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35:26337–26349, 2022.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 13 Oct. 2023.

Z. Lu, A. Zhou, K. Wang, H. Ren, W. Shi, J. Pan, M. Zhan, and H. Li. MathCoder2: Better math reasoning from continued pretraining on model-translated mathematical code. *arXiv preprint arXiv:2410. 08196*, 2024a.

Z. Lu, A. Zhou, K. Wang, H. Ren, W. Shi, J. Pan, M. Zhan, and H. Li. Step-controlled dpo: Leveraging stepwise error for enhanced mathematical reasoning. *arXiv preprint arXiv:2407.00782*, 2024b.

G. L. Marchetti, G. Cesa, K. Pratik, and A. Behboodi. Neural lattice reduction: A self-supervised geometric deep learning approach. *arXiv [cs.LG]*, 14 Nov. 2023.

M. Mikuła, S. Antoniak, S. Tworkowski, B. Piotrowski, A. Jiang, J. P. Zhou, C. Szegedy, Ł. Kuciński, P. Miłoś, and Y. Wu. Magnushammer: A transformer-based approach to premise selection. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023. URL https://openreview.net/forum?id=WgaVCqZeIU.

L. d. Moura and S. Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, Lecture notes in computer science, pages 625–635. Springer International Publishing, Cham, 2021.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. J. Lowe. Training language models to follow instructions with human feedback. *Neural Inf Process Syst*, abs/2203.02155:27730–27744, 4 Mar. 2022.

R. Y. Pang, W. Yuan, H. He, K. Cho, S. Sukhbaatar, and J. E. Weston. Iterative reasoning preference optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=4XIKfvNYvx.

L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Lecture Notes in Computer Science. Springer, Berlin, Germany, 1994 edition, 28 July 1994.

S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *arXiv [cs.LG]*, 2020.

S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

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, 2024.

E. Real, Y. Chen, M. Rossini, C. de Souza, M. Garg, A. Verghese, M. Firsching, Q. V. Le, E. D. Cubuk, and D. H. Park. AutoNumerics-zero: Automated discovery of state-of-the-art mathematical functions. *arXiv [cs.NE]*, 13 Dec. 2023.

P. Scholze. Liquid tensor experiment. *Exp. Math.*, 31(2):349–354, 3 Apr. 2022.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Z. Shao, F. Huang, and M. Huang. Chaining simultaneous thoughts for numerical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2533–2547, Stroudsburg, PA, USA, Dec. 2022. Association for Computational Linguistics.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv [cs.CL]*, 5 Feb. 2024.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

Y. Tong, X. Zhang, R. Wang, R. Wu, and J. He. DART-math: Difficulty-aware rejection tuning for mathematical problem-solving. *arXiv [cs.CL]*, 18 June 2024.

J. Uesato, N. Kushman, R. Kumar, F. Song, N. Siegel, L. Wang, A. Creswell, G. Irving, and I. Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv [cs.LG]*, 25 Nov. 2022.

H. Wang, H. Xin, C. Zheng, Z. Liu, Q. Cao, Y. Huang, J. Xiong, H. Shi, E. Xie, J. Yin, Z. Li, and X. Liang. LEGO-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*, Oct. 2023.

M. Wang and J. Deng. Learning to prove theorems by learning to generate theorems. *Advances in Neural Information Processing Systems*, 33:18146–18157, 2020.

P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Stroudsburg, PA, USA, 2024. Association for Computational Linguistics.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

S. Welleck and R. Saha. Llmstep: Llm proofstep suggestions in lean. *arXiv preprint arXiv:2310.18457*, 2023.

E. Wenger, M. Chen, F. Charton, and K. Lauter. SALSA: Attacking lattice cryptography with transformers. In *Advances in Neural Information Processing Systems*, 31 Oct. 2022.

T.-R. Wu, C.-C. Shih, T. H. Wei, M.-Y. Tsai, W.-Y. Hsu, and I.-C. Wu. AlphaZero-based proof cost network to aid game solving. In *International Conference on Learning Representations*, 6 Oct. 2021.

Y. Wu, A. Q. Jiang, W. Li, M. Rabe, C. Staats, M. Jamnik, and C. Szegedy. Autoformalization with large language models. *Neural Inf Process Syst*, abs/2205.12615:32353–32368, May 2022.

Y. Xie, A. Goyal, W. Zheng, M.-Y. Kan, T. P. Lillicrap, K. Kawaguchi, and M. Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

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. *arXiv preprint arXiv:2405.14333*, 2024a.

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. *arXiv [cs.CL]*, Aug. 2024b.

W. Xiong, C. Shi, J. Shen, A. Rosenberg, Z. Qin, D. Calandriello, M. Khalman, R. Joshi, B. Piot, M. Saleh, C. Jin, T. Zhang, and T. Liu. Building math agents with multi-turn iterative preference learning. *arXiv [cs.LG]*, 3 Sept. 2024.

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.

L. Yuan, G. Cui, H. Wang, N. Ding, X. Wang, J. Deng, B. Shan, H. Chen, R. Xie, Y. Lin, et al. Advancing llm reasoning generalists with preference trees. In *AI for Math Workshop@ ICML*, 2024.

X. Zhao, W. Li, and L. Kong. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv [cs.CL]*, May 2023.

X. Zhu, J. Wang, L. Zhang, Y. Zhang, Y. Huang, R. Gan, J. Zhang, and Y. Yang. Solving math word problems via cooperative reasoning induced language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, July 2023.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv [cs.CL]*, 18 Sept. 2019.

A Extended Discussions on Related Works

Reference	Verifier	Task	Output	Premise Retrieval	Proof Search
GPT-f (Polu and Sutskever, 2020)	Metamath	Prover	Proof Step	-	Best First Search
Lean-Gym (Polu et al., 2023)	Lean	Prover	Proof Step	-	Best First Search Expert iteration
LeanDojo (Yang et al., 2023)	Lean	Prover	Proof step	RAG Offline Database	Best First Search
LeanAgent (Kumarappan et al., 2024)	Lean	Prover	Proof step	RAG Online Database	Best First Search
HTPS (Lample et al., 2022)	Lean Metamath	Prover	Proof step	-	HTPS
DeepSeek-Prover (Xin et al., 2024a)	Lean	Prover Reasoning	Whole Proof	-	-
DeepSeek-Prover 1.5 (Xin et al., 2024b)	Lean	Prover Reasoning	Proof Step Whole Proof	-	MCTS
LLEMA (Azerbayev et al., 2024)	Any Tool	Prover Reasoning	Proof Text	-	Best First Search
Verify Step by Step (Lightman et al., 2023)	ORM-PRM	Reasoning	Text	-	Best First Search

Table 3: A summary of models for theorem proving and mathematical reasoning.

Theorem Proving. Machine learning is currently used in automatic theorem proving to generate the whole proof, or provide assistance in sequential proofs by finding the right premise or right tactics, or assisting in theorem prover specific formalization of mathematical statements, as well as combination of these ideas; see (Xin et al., 2024a;b; Polu and Sutskever, 2020; Jiang et al., 2022a; Wang et al., 2023; Jiang et al., 2022b; Wu et al., 2022) for some pointers. We review some of these works in more details. A key component for theorem proving is the formal proof management systems and verifiers like Lean (Moura and Ullrich, 2021) and Coq (Coq Development Team, 2024). Lean has received particular attention in the mathematics community for example by its use in verification of some components in the condensed mathematics research program (Scholze, 2022). We use Lean in this work, and a framework that consists of a premise retrieval part and a prover that generates step-wise proof tactics.

Machine learning can be used to find useful premises, definitions and theorems (Irving et al., 2016). For Coq framework (Coq Development Team, 2024), the authors in Blaauwbroek et al. (2024) introduce a graph neural network for embedding the new definitions using a hierarchical representation based on graph representations of theorems and definitions in the Tactician platform (Blaauwbroek, 2024). This enables them to use recent proofs and theorems in Coq, while kNN is used for the more recent tactics written by the user. In Thor (Jiang et al., 2022a), the authors introduce a class of methods to use automated theorem provers for premise selection. LeanDojo (Yang et al., 2023) provided a framework that retrieves the premise from a database of Lean premises and uses ReProver to generate tactics for theorem proving in Lean. LeanAgent (Kumarappan et al., 2024) adds a dynamic database to LeanDojo, which enables the continual learning of the agent. Besides, a new curriculum learning based on the difficulty of the theorems is also added.

Searching over different proofs and tactics is another component explored in various works, for example (Polu et al., 2023; Xin et al., 2024b; Lample et al., 2022). As discussed in Polu et al. (2023), any RL algorithm for theorem proving should address two challenges, namely an infinite dimensional action space, and the absence of a natural opponent for self-play. Therefore, the authors suggest using expert iteration (Anthony et al., 2017), which amounts to iteratively fine-tuning a based model and searching to generate correct proofs. They also introduce `lean-gym` to facilitate the search procedure in Lean. HTPSs was introduced in Lample et al. (2022) which is a new search algorithms within an online reinforcement learning procedure. The key idea is that the search is represented over a hypergraph, where a policy network generates tactics composed of sub-goals, and then each

goal is expanded for proof using new tactics. The provability of each goal is approximated using a critic. The idea of keeping the visit counts, action values and their statistics follows the similar MCTS procedure. In Gloeckle et al. (2024), the authors introduce a reinforcement learning based framework for theorem proving in Lean 4, which consist of using a programming interface based on Aesop for proof search organization, the HTPS ((Lample et al., 2022)) procedure with an online reinforcement learning step. In Zhao et al. (2023), the authors use subgoal learning from reinforcement learning to decompose an informal LLM generated proof into subgoals and verify using a verifier. They also use a diffusion model for demonstration organization.

The transformer based language models are used also for theorem proving. For example GPT-*f* was introduced in Polu and Sutskever (2020) using Metamath as the formalization framework. Besides, they showed that iterative training a value functionon proofs generated by the prover can continually improve the performance. Another example is LLEMMA which is based on pretrained Llama Code (Azerbayev et al., 2024).

The proofs in some frameworks are already in the language of formal theorem provers. Besides, the external tools for mathematical reasoning can include codes for mathematical arguments. Machine learning has been used to help the formalization of mathematical statements suitable for the automatic theorem provers (Wu et al., 2022; Jiang et al., 2022b; Zhao et al., 2023). Furthermore, there is a spectrum of methods that starts here with methods operating in a formal language and ends with *informal* proofs and solutions in natural language. In MathCoder2 Lu et al. (2024a), the authors provided pairs of mathematical codes and the associated natural language versions.

DeepSeek-Prover (Xin et al., 2024a) generated Lean4 proof data by translating natural language problems into their formal versions in Lean4. The model produces the whole proof in a single turn. DeepSeekV1.5 (Xin et al., 2024b) provides a middle ground by generating the whole proof, verifying it via Lean, and then truncating the proof until the first error. The authors propose additional techniques relying on proper appending of previous states and including truncate-and-resume in the MCTS procedure. They also leverage Verification feedback from Lean and improve model's alignment with the formal structure of Lean. The authors in Wang et al. (2023) provide another hybrid solution where first an informal proof is generated, then broken into sub-components, and then a lemma is retrieved from a library of lemmas considered as skills, and finally the final formalized proof is generated using the previous components. They use the framework of Isabelle (Paulson, 1994) for theorem proving.

Treating mathematical problem solving as a reasoning task has been considered in many works, for example (Tong et al., 2024; Zhu et al., 2023; Shao et al., 2022; 2024; Guan et al., 2025). Among them, there are models, like Shao et al. (2024), based on chain-of-thought (Wei et al., 2022) or program of thought (Gao et al., 2023; Chen et al., 2022). The generator-verifier configuration has been considered in Zhu et al. (2023), where they train a step and path verifier for reasoning. Integrating a tool for improving reasoning of large language models has been used in Gou et al. (2023). In contrast to our work, these models do not rely on theorem provers. The authors in Wang and Deng (2020) address the lack of human labeled theorems and proofs to use for supervised training by training a generative model and, then, use the synthesized pairs to train a theorem prover model.

The idea of proof size and using it for guiding the search has been discussed in Wu et al. (2021); Polu et al. (2023). There is a line of work related to discovering new mathematical functions or solving mathematical problems, see for example (Real et al., 2023; Alfarano et al., 2024; Marchetti et al., 2023; Wenger et al., 2022). However, these works do not rely on proof verifiers and fall outside the scope of this work.

Preference Optimization. Another important step in theorem proving, and many other alignment related tasks, is to *align* the output of a given model based on the positive-negative preference pair of samples. There are online version of RLHF as in Bai et al. (2022); Ouyang et al. (2022), and offline versions with either an explicit reward model (e.g. (Christiano et al., 2017; Ziegler et al., 2019)) or an implicit reward model (e.g. DPO (Rafailov et al., 2024)) to encode the preference. The later method directly optimizes the model without training any independent reward model. In particular DPO uses Bradley-Terry preference model and the fact that the optimal solution to the KL-constrained reward maximization objective is known in closed form to simplify the training loss without dependence on a reward model (Rafailov et al., 2024). Many follow-up works explored DPO variations and its shortcomings such as reward over-optimization (Gorbatovski et al., 2024; Ethayarajh et al., 2024).

Different forms of direct preference optimization have been explored in the recent literature for enhancing the reasoning capabilities of language models, in particular for mathematical problem solving, for example (Xiong et al., 2024; Yuan et al., 2024; Jiao et al., 2024; Cobbe et al., 2021; Lightman et al., 2023; Wang et al., 2024; Pang et al., 2024; Chen et al., 2024; Lu et al., 2024b). The authors in Xiong et al. (2024) introduce a multi-turn version of DPO to use feedbacks from the verifiers, in their case the code interpreters, particularly for multi-turn scenarios, which requires trajectory-level preference optimization. The trajectory-level preference can be obtained by dataset labels of the gold answers or ORMs (Cobbe et al., 2021; Lightman et al., 2023). A more fine-grained step-wise supervision can be used as in PRMs (Lightman et al., 2023) or by leveraging trajectory level preferences (Wang et al., 2024). The idea of ORM and PRM has also been originally discussed in Uesato et al. (2022). These works focus on mathematical reasoning, rather then formal theorem proving, which is the focus of our work. In our work, the theorem prover automatically provides the preference, which is used either during training or for an offline generated dataset. Furthermore, our work follows PRM philosophy as we consider feedback from Lean at each step of proof generation.

The authors in Jiao et al. (2024); Yuan et al. (2024) applied the original DPO or KTO by taking trajectory completion as a meta action. The online iterative versions of DPO originally designed for chat is adapted to achieve better CoT reasoning in Pang et al. (2024); Xie et al. (2024). In the papers (Chen et al., 2024; Lai et al., 2024; Xie et al., 2024; Lu et al., 2024b), the authors have explored generating proxy step-wise labels for the intermediate steps of the reasoning trajectories.

B Additional Details

Algorithm 1 DPO dataset generation using the **zero accuracy** strategy

Input: training dataset D_{train}, reference policy π_{ref}, retrieval model `retriever`
$D \leftarrow [\,]$
for theorem and g.t. proof $(T, P) \in D_{\text{train}}$ **do**
  for proof state and g.t. tactic $(s_t, \widehat{a}_t) \in P$ **do**
    $p_t \leftarrow \texttt{retriever}(s_t)$ // compute the premises for retrieval-augmentation
    $A_t \leftarrow \{a_t^j \sim \pi_{\text{ref}}(a_t|s_t, p_t) \mid j = 1, \ldots, 8\}$ // with *beam-search* (beam-width $w = 8$)
    sort(A_t) // by decreasing $\pi_{\text{ref}}(a|s_t, p_t)$ (if needed)
    if $\widehat{a}_t \notin A_t$ **then**
      append $\widehat{a}_t$ at the end of A_t ($w = 9$) // add the ground truth tactic if needed
    end if
    $A_t^+ \leftarrow \{a_t^j \mid \text{Lean}(a_t^j|s_t, T) = +\}$ // gather Lean feedback for each sampled tactic
    $A_t^- \leftarrow \{a_t^j \mid \text{Lean}(a_t^j|s_t, T) = -\}$
    for $y^- \in A_t^-$ **do**
      for $y^+ \in A_t^+$ **do**
        if $\pi_{\text{ref}}(y^-|s_t, p_t) > \pi_{\text{ref}}(y^+|s_t, p_t)$ **then**
          $x \leftarrow \text{dynamic\_prompt}(s_t, p_t)$ // prompt augmentation by premises dropout
          D.append((x, y^+, y^-))
          move y^+ at the end of A_t^+ to lower its priority // simple heuristic to avoid oversampling y^+
          break // exit the inner loop over A_t^+ and move to the next y^-
        end if
      end for
    end for
  end for
end for
Output: D