HyperTree Proof Search for Neural Theorem Proving

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose an online training procedure for a transformer-based automated theorem prover. Our approach leverages a new search algorithm, HyperTree Proof Search (HTPS), inspired by the recent success of AlphaZero. Our model learns from previous proof searches through online training, allowing it to generalize to domains far from the training distribution. We report detailed ablations of our pipeline's main components by studying performance on three environments of increasing complexity. In particular, we show that with HTPS alone, a model trained on annotated proofs manages to prove 65.4% of a held-out set of Metamath theorems, significantly outperforming the previous state of the art of 56.5% by GPT-f. Online training on these unproved theorems increases accuracy to 82.6%. With a similar computational budget, we improve the state of the art on the Lean-based miniF2F-curriculum dataset from 31% to 42% proving accuracy.

1 Introduction

The nineteenth century saw the emergence of proofs so involved that they could only be verified by a handful of specialists. This limited peer review process inevitably led to invalid proofs, with mistakes sometimes remaining undiscovered for years (e.g. the erroneous proof of the Four Colour Conjecture [1]). Some mathematicians argue that the frontier of mathematics has reached such a level of complexity that the traditional review process is no longer sufficient, envisioning a future where articles are submitted with formal proofs so that the correctness can be delegated to a computer [2].

Unfortunately, very few mathematicians have adopted formal systems in their work, and as of today, only a fraction of existing mathematics has been formalized. Several obstacles have hindered the widespread adoption of formal systems. First, formalized mathematics are quite dissimilar from traditional mathematics, rather closer to source code written in a programming language, which makes formal systems difficult to use, especially for newcomers. Second, formalizing an existing proof still involves significant effort and expertise (the formalization of the Kepler conjecture took over 20 person years to complete [3]) and even seemingly simple statements sometimes remain frustratingly challenging to formalize.

To write a formal proof, mathematicians typically work with Interactive Theorem Provers (ITPs). The most popular ITPs provide high-level "tactics" that can be applied on an input theorem (e.g. the initial goal) to generate a set of subgoals, with the guarantee that proving all subgoals will result in a proof of the initial goal (reaching an empty set means the tactic solves the goal). An example of a proof in Lean [4], an interactive theorem prover, is given in Figure 1 and the corresponding proof hypertree[1] is illustrated in Figure 5 of the Appendix. A tactic, `induction k`, is applied on the root goal ($n + k \leq m + k$) to start a proof by induction. The formal system returns two subgoals: $n + 0 \leq m + 0$ (the initial case) and $n + k \leq m + k \Rightarrow n + k + 1 \leq m + k + 1$ (the induction step).

[1] A hypergraph is a graph where an edge leads to a set of nodes that is potentially empty in our set-up. A hypertree, in this work, is a hypergraph without cycles. Formal definitions can be found in Appendix C.1

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

As the first subgoal is our initial hypothesis, it can be solved using the `exact` tactic. To prove the second subgoal, we first rewrite it using `nat.succ_le_succ_iff`, a theorem from the Lean library stating that $m + 1 \leq n + 1 \iff m \leq n$.

The new subgoal then becomes our induction hypothesis $n + k \leq m + k$, and can again be solved using the `exact` tactic, thereby solving the last remaining subgoal and completing the proof of the initial statement. Starting from a goal and reducing it to subgoals until these can be solved is commonly referred to as backward proving.

```
import data.nat.basic
example (m n k : ℕ) (h₀ : n ≤ m) : n + k ≤ m + k := begin
  induction k,
  {
    exact h₀
  },
  {
    rw nat.succ_le_succ_iff,
    exact k_ih
  }
end
```

First subgoal : $n + 0 \leq m + 0$

Second subgoal : $n + k \leq m + k \Rightarrow n + k + 1 \leq m + k + 1$

Figure 1: **A simple proof of the statement $\mathbf{n \leq m \Rightarrow n + k \leq m + k}$ in Lean.** The *induction* tactic reduces the initial statement to two subgoals, that can be solved independently.

In this paper, we aim at creating a prover that can automatically solve input theorems by generating a sequence of suitable tactics without human interaction. The backward procedure naturally suggests a simple approach where a machine learning model trained to map goals to tactics interacts with an ITP to build the proof of an input goal in a backward fashion. The automated prover builds a hypergraph with the theorem to be proved as the root node, tactics as edges and subgoals as nodes. The prover recursively expands leaves by generating tactics with our model until we find a proof of the initial theorem. A proof is then a hypertree rooted in the initial theorem whose leaves are empty sets. Search in this setting is analog to search in two player games.

However, particular challenges arise in theorem proving. First, the action space, i.e. the amount of possible "moves" in a given state, is infinite (there is an unlimited number of tactics that can be applied to a given theorem). This requires sampling possible actions from a language model for which training data is scarce. Moreover, if all tactics sampled at a goal fail, we have no information on what region of the probability space to sample next. Second, in the context of theorem proving, we need to provide a proof of all subgoals created by a tactic, whereas AlphaZero[5] for two player games is allowed to focus on the most likely adversary moves. Third, in Chess or Go, playing a sub-optimal move does not necessarily lead to losing the game, thus exploring these branches can provide information. In theorem proving, it is frequent to generate tactics that result in subgoals that can no longer be proved and on which the model will waste significant search budget.

This paper presents an in-depth study of our approach to overcome these difficulties and the resulting model, Evariste. In particular, we make the following contributions:

- A new MCTS-inspired search algorithm for finding proofs in unbalanced hypergraphs.
- A new environment (Equations) to easily prototype and understand the behavior of the models we train and our proof search.
- A detailed ablation study and analysis of the different components used in our approach on three different theorem proving environments. We study how data is selected for training the policy model after a successful or failed proof-search, what target should be used to train the critic model, and the impact of online training vs. expert iteration.
- State-of-the-art performance on all analyzed environments. In particular, our model manages to prove over 82.6% of proofs in a held-out set of theorems from `set.mm` in Metamath, as well as 58.6% on miniF2F-valid [6] in Lean.

We begin by introducing related work in Section 2 and present the three theorem proving environments that we study in Section 3. Then, we present our method in Section 4, and all experimental details in Section 5. Finally, we describe our main results and ablation studies in Section 6.

2 Related work

Automated theorem proving has been a long-standing goal of artificial intelligence, with the earliest work dating from the 1950s [7, 8]. We focus here on recent work closest to ours and defer additional related work to Appendix B.

Neural theorem provers. Recent work applying deep learning methods to theorem proving [9-11] are the closest to this work and obtained impressive results on difficult held-out sets for Metamath and Lean. The main differences between their approach and ours are the proof-search algorithm we propose, the training data we extract from proof-searches and our use of online training compared to their expert iterations. Another similar approach is Holophrasm [12], which is based on a different tree exploration technique. Unlike previous studies that focus on a single proving environment (e.g. Metamath, Lean, or HOL-Light), we extensively study the performance of our prover on three different formal languages, and found that some conclusions significantly vary based on the considered environment.

MCTS and two player games. AlphaZero [5] demonstrated good performances on two player games, replacing the Monte-Carlo evaluations of MCTS [13] with evaluations from a deep neural network and guiding the search with an additional deep policy. This recent success follows extensive literature in search methods for two player games, notably alpha-beta search [14]. Theorem proving can be thought of as computing game-theoretic value for positions in a min/max tree: to prove a goal, we need one move (max) that leads to subgoals that are all proven (min). Noticing heterogeneity in the arities of min or max nodes, we propose a search method that goes down simultaneously in all children of min nodes, such that every simulation could potentially result in a full proof-tree.

3 Proving environments

In this paper, we develop and test our methods on three theorem proving environments: Metamath, Lean and Equations. Metamath [15] comes with a database of $30k$ human-written theorems called `set.mm`. We also evaluate our methods on the Lean proving environment, which provides a level of automation that is helpful to solve more complex theorems. Lean comes with a human-written library of $27k$ theorems called Mathlib [16]. Finally, since Metamath proofs can be quite difficult to understand and Lean requires more computing resources, we developed our own environment called Equations, restricted to proofs of mathematical identities. Its simplicity makes it ideal for prototyping and debugging. We briefly introduce these environments below.

3.1 Metamath

Metamath's only rule is string substitution. Starting from a theorem to be proven, variables are substituted until we reach axioms. In our setup, we consider a tactic to be the label of a theorem in `set.mm`, along with the necessary substitutions. For instance, to show that $2 + 2 = 4$, we can use the rule `eqtr4i` which states that $A = B \wedge C = B \Rightarrow A = C$ with substitutions: $A = (2 + 2)$, $B = (2 + (1 + 1))$, and $C = 4$. We are then left with two subgoals to prove: $(2 + 2) = (2 + (1 + 1))$ and $4 = (2 + (1 + 1))$. We give the corresponding proof-tree in Figure 4 of the Appendix.

The simplicity of Metamath makes it a great test bed for our algorithms. However, its lack of automation leads to larger proof sizes and its syntax and naming conventions make each step difficult to interpret for neophytes. Similar to GPT-f[9], we implement a parser for Metamath in order to automatically prove the syntactic correctness of statements. Moreover, we use this parser to allow generating only substitutions that cannot be inferred from the goal.

3.2 Lean

Lean is a full-fledged programming language and benefits from more powerful automation than Metamath, with tactics such as `ring` (able to prove goals using manipulations in semirings), `norm_num` (able to prove numerical goals) or `linarith` (able to find contradictions in a set of inequalities). Unlike Polu and Sutskever [9], our Lean API attempts to split tactic states into separate subgoals when no metavariable is shared. More details about our API and an example proof-tree can be found in Appendix D.5 and Figure 5 of the Appendix.

3.3 Equations

We developed the Equations environment as a simpler analogue to existing proving environments. Its expressivity is restricted to manipulating mathematical expressions (e.g. equalities or inequalities) with simple rules (e.g. $A + B = B + A$, or $A < B \Rightarrow -B < -A$). This reduced expressivity makes goals and tactics easy to understand, helping with interpretability and debugging: plotting the set of goals explored during a Metamath proof search does not give a lot of insights on whether it is on track to find a proof. In Section E of the appendix, we give an in-depth presentation of this environment.

Unlike in Metamath or Lean, we do not have access to a training set of human annotated proofs for this environment. Instead, we create a training set composed of randomly generated synthetic theorems and their proofs (see Section E.5 for details), and manually create an out-of-domain set of non-trivial mathematical identities for which we do not provide proofs. We refer to this evaluation split as *Identities*, a set of 160 mathematical expressions. As synthetic theorems randomly generated are much simpler and significantly differ from statements in the *Identities* split, we can evaluate the ability of our model to generalize to complex and out of domains data. An example proof-tree in Equations is shown in Figure 6 of the Appendix.

4 Online training from proof searches

In this section, we introduce our Hypertree Proof Search (HTPS) algorithm and describe how it is used to generate training data for our model. We then detail our online training method.

4.1 Hypertree Proof Search

Given a main goal g to automatically prove, HTPS is the algorithm that interacts with a policy model P_θ and a critic model c_θ, and the theorem proving environment to find a proof hypertree for g. Proof search progressively grows a hypergraph starting from g, iteratively repeating the three steps illustrated in Figure 2: selection, expansion and back-propagation. The main difference with other search algorithms is our parallel descent in all subgoals of a tactic. A proof is found when there exists a hypertree from the root to leaves that are empty sets. We explain HTPS in full details in Appendix C and compare it to other search methods in Appendix C.5.

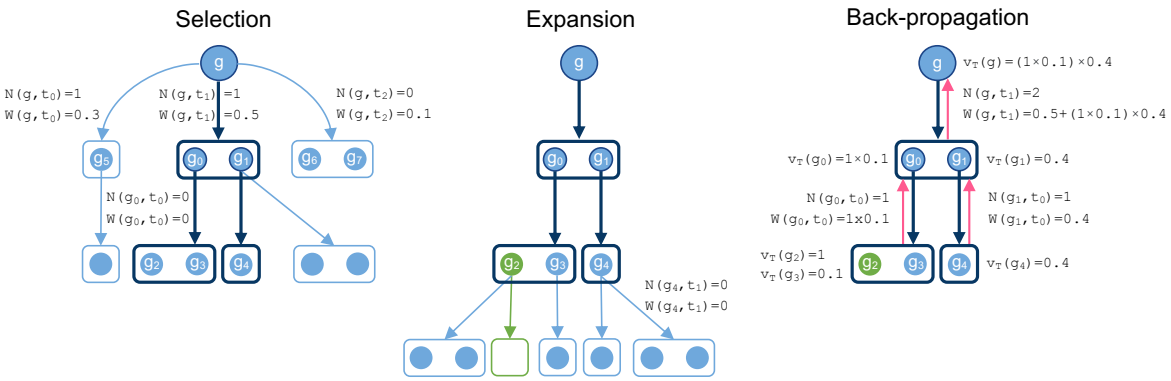

Figure 2: **HyperTree Proof Search**. We aim at finding a proof of the root theorem g. The figure represents the three steps of HTPS that are repeated until a proof is found. Proving either $\{g_5\}$, $\{g_0, g_1\}$, or $\{g_6, g_7\}$ would lead to a proof of g by tactic t_0, t_1, or t_2. Guided by the search policy, we select a hypertree whose leaves are unexpanded nodes. Selected nodes are then expanded, adding new tactics and nodes to the hypergraph. Finally, we evaluate the node values v_T of the hypertree starting from the leaves, using the critic, and back-propagating to the root, updating the visit counts N and total action values W.

4.2 Online training

Both the policy model P_θ and the critic model c_θ are encoder-decoder transformers [17] with shared weights θ, which are trained online on two different objectives. The policy model P_θ takes as input a tokenized goal and generates tactics. It is trained with a standard seq2seq objective [18], where we minimize the cross-entropy loss of predicted tactic tokens conditioned on the input goal.

Our critic model c_θ is used to predict floating point values representing how likely a goal is to be solved. We start decoding with a special token, restrict the output vocabulary to two tokens PROVABLE and UNPROVABLE, and evaluate the critic with $c_\theta(g) = P(\texttt{PROVABLE}|g, \texttt{CRITIC})$. This objective is identical to a seq2seq objective where the cross-entropy is minimized over the two special tokens.

Our online training uses a distributed learning architecture reminiscent of AlphaZero [19] or distributed reinforcement learning setups [20, 5]. A distributed data parallel trainer receives training data from a set of asynchronous provers that run proof searches on theorems chosen by a controller. Provers, in turn, continuously retrieve the latest model versions produced by the trainers in order to improve the quality of their proof search. This set-up is represented in Figure 7 of the Appendix. Once a prover finishes a proof-search, we extract two types of training samples from its hypergraph:

Tactic samples. At the end of a successful proof search, we extract (goal, tactic) pairs of a minimal proof hypertree of the root node as training samples for the policy model. We show that this selection has a large impact on performances, other options such as selecting all solved nodes are investigated in Section 6.2.1. We use a different minimality criteria depending on the environment: number of proof steps for Metamath and Equations and total tactic CPU time for Lean. It is trained with a standard seq2seq objective [18], where we minimize the cross-entropy loss of predicted tactic tokens conditioned on an input goal.

Critic samples. In the proof search hypergraph, we select all nodes that are either solved, invalid (all tactics failed or led to invalid nodes), or with a visit count higher than a threshold. Then, we use $c(g) = 1$ as the training target for solved nodes. For internal nodes, we use the final estimated action value $c(g) = W(g, t^*)/N(g, t^*)$ where t^* is the tactic that maximizes the search policy at g. Finally, for invalid nodes, we use $c(g) = 0$.

The trainers receive training samples that are stored into two separate finite-size queues, one for each objective. When a queue is full, appending a new sample discards the oldest one. In order to create a batch for a task, we uniformly select samples in the corresponding queue. During online training, we continue sampling from the supervised tasks which provide high-quality data. All training objectives are weighted equally. An overwiew of our full training pipeline is given in Appendix D.1.

Our proof-search depends on many hyper-parameters, and the optimal settings might not be the same for all statements, making tuning impractical. Thus, the controller samples these hyper-parameters from pre-defined ranges (see Appendix D.3 for details) for each different proof-search attempt.

5 Experiments

In this section, we provide details about our experimental training and evaluation protocols. We first describe the supervised datasets used to fine-tune our policy models, as well as the tokenization used. Then, we discuss the evaluation datasets and methodology. In Appendix D.2, we provide additional details about our model pretraining and architecture.

5.1 Model fine-tuning and supervised datasets

Starting the HTPS procedure described in Section 4 from a randomly initialized model would be sub-optimal, as no valid tactic would ever be sampled from the policy model. Thus, starting the online training from a non-trivial model is critical. To this end, we first fine-tune our policy model P_θ on a supervised dataset of theorems specific to each environment.

Metamath In Metamath, we extract all proofs from the `set.mm` library, composed of 37091 theorems (c.f. Section D.6 for the version of `set.mm`). We first derive a graph of dependencies between statements, and generate a random train-valid-test split of theorems, with 1000 valid and test theorems. We use the DAG to ensure that each theorem in the valid or test set is not used to prove another theorem. Moreover, this DAG is used to build a table of *forbidden tokens*: if the proof of A depends on B, we set to zero the probability of generating the token A during a proof-search of B. More details on our seq2seq training objective and co-training tasks can be found in Appendix D.4. The training set is composed of around 1M goal-tactic pairs; more statistics about the training data are provided in Table 1. Tokenization in Metamath is trivial, as statements are composed of space-separated tokens.

Table 1: **Dataset statistics for supervised training.**

	# train theorems	# train proof steps	Avg. goal length
Equations	∞	∞	33.7
Metamath	35k	1M	120.1
Lean	24k	144k	169.3

Lean Following [11], we extract a supervised dataset from the Mathlib library. The training set is composed of 24k theorems and 144k goal-tactic pairs. In addition, we co-train with the dataset of proof-artifacts of Han et al. [10] to reduce overfitting. To facilitate experimentation and reproducibility, we use fixed versions of Lean, Mathlib, and miniF2F (c.f. Appendix D.6). Finally, we add another supervised co-training task by converting to Lean a synthetic dataset of theorems generated by the Equations environment (c.f. Appendix E.6). In order to avoid hooking into the Lean parser, we tokenize goals and tactics using byte-pair encoding (BPE [21]) following previous work [9, 11]. Statistics about the training set are available in Table 1.

Equations Unlike Metamath or Lean, the Equations environment does not come with a dataset of manually annotated proofs of theorems. Instead, we generate supervised data on the fly using the random graph generator described in Appendix E.5. As the model quickly reaches a 100% proving accuracy on these synthetic theorems, there would be no benefit in using them during online training. Thus, we fine-tune on the synthetic dataset, and only leverage statements from the *Identities* split during online training. As in Metamath, tokenization of statements for this environment is natural, as each statement can be tokenized using the list of symbols from its prefix decomposition [22].

5.2 Evaluation settings and protocol

In Polu et al. [11], the model is fine-tuned on theorems from the training set and expert iteration is done on theorems from different sources: train theorems, synthetic statements, and an extra curriculum of statements without proofs (miniF2F-curriculum). The produced model is then evaluated on unseen statements, namely the validation and test splits of the miniF2F dataset [6].

In this work, we also consider the *transductive* setup: on a corpus of unproved statements available at train time, how many proofs can our method learn to generate? This protocol is also sensible, as allowing the model to learn from a failed proof-search can lead to more focused exploration on the next attempt, proving more statements overall than a model that would not be trained online.

Following [9], we also evaluate the pass@k by running k proof searches on the evaluated statements with the policy and critic obtained by online training. Given the many evaluations presented in this work, we only run them once. We give more details on the hyper-parameters used in Appendix D.3. In the transductive setup, we also report the *cumulative pass rate*, i.e. the proportion of theorems solved at least once during online training.

6 Results

In this section, we present our results and study the moving parts of our pipeline through ablations. We compare our results with GPT-f which represents the state of the art on Metamath and Lean.

Table 2: **Pass rate on Lean environment using 64 trials (pass@64)** Numbers with a † exponent correspond to the cumulative pass-rate since the evaluated statements are part of the online training.

	Supervised	GPT-f	Evariste-1d	Evariste-7d	Evariste
Online training statements	-	miniF2F-curriculum			miniF2F-valid
miniF2F-valid	38.5	47.3	46.7	47.5	58.6†
miniF2F-test	35.3	36.6	38.9	40.6	41.0
miniF2F-curriculum	20.8	30.6	33.6†	42.5†	32.1
Train time (A100 days)	50	2000	230	1620	1360

6.1 Main results

6.1.1 Lean

In Lean, we run our experiments on A100 GPUs with 32 trainers and 200 provers. Each prover runs our Lean API on 48 CPU cores. Unlike Polu et al. [11], we sample statements equally from mathlib-train and miniF2F-curriculum, to avoid giving too much importance to statements from a different domain than the target. Results can be found in Table 2. After 1 day of training, each statement from miniF2F-curriculum has been sampled on average 250 times, and 110 out of the 327 statements have been solved. Our model outperforms GPT-f on miniF2F-test, with an approximately $10\times$ training time speed-up. After 7 days, we solve 139 statements of miniF2F-curriculum (100 for GPT-f), and observe further improvements on miniF2F-valid or miniF2F-test.

For other evaluations, we depart from the set-up of Polu et al. [11], directly using the statements from the miniF2F-valid split in our online training, obtaining 58.6% cumulative pass rate. We then evaluate the final model on miniF2F-test, reaching 41% pass@64, against 36.6% for GPT-f. Without the synthetic data co-training task, the performance drops to 54.9% cumulative pass rate on the miniF2F-valid split, and 38.5% pass@64 on the miniF2F-test split. Examples of proofs found by our model can be found in Appendix F.

6.1.2 Metamath

On Metamath, we train our model on V100 GPUs, with 128 trainers and 256 provers, whereas ablations are run on 16 trainers and 32 provers. We report our results in Table 3 for the supervised model and for a model trained with online training. During online training, we sample statements from the training and from the validation splits of `set.mm` equally.

Online training dramatically improves performances on valid statements, going from a 61% pass@8 to a cumulative pass rate of 82.6%. This improvement cannot solely be explained by the high number of attempts on validation theorems during training. Indeed, the ablation in Figure 3 (right) shows that Evariste significantly outperforms a supervised model with the same number of attempts. The supervised model plateaus at 66% while Evariste keeps improving beyond 74% after 7 days of training, showing that the model is able to learn from previous proof searches through online training.

On test theorems, for which statements were not provided during online training, the accuracy increased by 10% compared to the supervised model, from 55.8% to 65.6% accuracy. The supervised model obtains a pass@32 accuracy of 65.4% (resp. 61.2%) on the validation (resp. test) splits, compared to GPT-f's 56.5% (resp. 56.2%) after expert iteration.

Table 3: **Results on Metamath for a supervised model and Evariste.** We report the pass@8 and pass@32 scores on the validation and test splits. Additionally, for Evariste we also report the cumulative score on the validation set, i.e. the fraction of theorems proved at least one time during online training. Note that for Evariste on Valid, the cumulative and pass@k performances are close since these statements were seen during training.

	Valid			Test	
	cumulative	pass@8	pass@32	pass@8	pass@32
Supervised	N/A	61.0%	65.4%	55.8%	61.2%
Evariste	82.6%	81.0%	81.2%	65.6%	72.4%

6.1.3 Equations

In Equations, we run our main experiment with 32 trainers and 64 provers, whereas ablations are run on 16 trainers and 32 provers. In this environment, the model easily learns the training distribution of our random generator, and solves all synthetically generated problems. Thus, online training is run on the *Identities* statements only. Our main experiment reaches a cumulative pass rate of 91.3% on the *Identities* split, while a supervised model never exceeds 36% even after a similar number of proof attempts. In Appendix 9, we give examples of *Identities* statements proved during online training, as well as the size and depth of proofs found by the model.

In particular, Evariste managed to find the proof of complex mathematical statements, such as $\sinh(x/2) = \sinh(x)/\sqrt{2(1+\cosh(x))}$ and $\tan(3x)(1-3(\tan(x))^2) = 3\tan(x) - (\tan(x))^2\tan(x)$ that required 82 and 117 proof steps respectively, showing the abilities of HTPS to

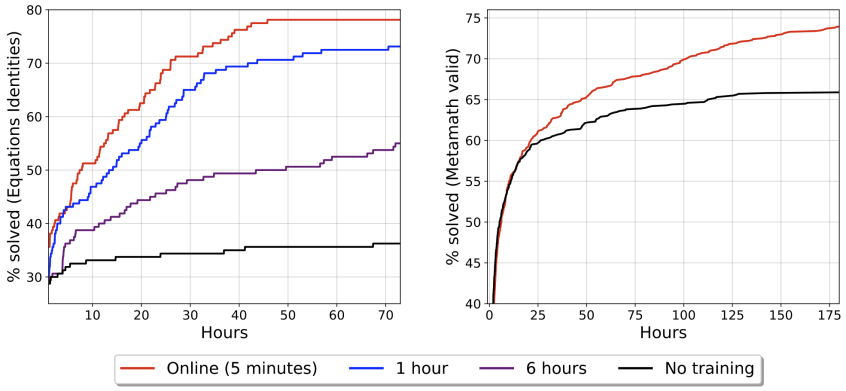

Figure 3: **Comparison between online setup, expert iteration, and fixed model.** We report the cumulative pass rate on the *Identities* (resp. valid) split on Equations (resp. Metamath). Reloading the model more frequently converges faster and to a better performance. When no training is done, the final performance is much lower despite using as many attempts, showing that online training is able to learn from previous proof searches.

prioritize subgoals and guide the search in very large proof graphs. This shows that online training is able to adapt our policy and critic models to a completely new domain, going from automatically generated statements to identities found in math books. Examples to understand the gap between these two domains can be found in Appendix E.

6.2 Ablation study

In this section, we present an ablation study on several components of our system. Since Lean experiments are CPU intensive, we run most of our ablations on the Equations and Metamath environments. On Lean, we ran experiments on a smaller subset of hyper-parameters that consistently performed well on the other environments.

6.2.1 Online training data for tactic objective

Table 4: **Performance of our model for different online training data for tactic objective.** We report the pass@8 score for Metamath and cumulative pass rate for Equations. We either keep all nodes and sample tactics according to the policy, or, extract (minimal) proofs of solved nodes, or (minimal) proofs of the root theorem only. Selecting minimal proofs always improves performance.

Proof Of	All Solved		Root		All Nodes
Type of Proof	All	Min	All	Min	
Metamath (valid)	61.2	65	57.4	**68.6**	51.6
Metamath (test)	57.2	**58.8**	54.8	57.4	54.4
Equations (Identities)	40.6	**78.1**	37.5	71.3	37.5

The way we filter tactics sent to the trainers has a large impact on final performances. We investigated several filtering methods and report the results in Table 4. The first method is similar to the one used in AlphaZero and exposed in [19]: we select all nodes of the proof search hypergraph where the visit count is above a certain threshold and we filter tactics above a given search policy score. At training time, tactics are sampled according to the filtered search policy. With this method the model reaches 51.6% pass@8 on the valid set of Metamath and 37.5% cumulative pass rate on Equations.

We then experimented with other filtering criteria, selecting only goal-tactic pairs that are part of proofs: either a proof of the root node, or of any solved node in the hypergraph. Then, we learn from all possible proofs, or only from proofs that are minimal according to a criteria (number of proof steps for Equations and Metamath, cumulative CPU time for Lean).

Learning only from minimal proofs always leads to improved performance, regardless of the selected roots. Learning from the minimal proofs of all solved nodes, we reach a cumulative pass rate of 78.1% on Equations, compared to 40.6% when learning from all proofs. On Metamath, only learning from the root's minimal proof gives the best result on the valid set, reaching a pass@8 of 68.6%.

6.2.2 Critic

Table 5: **Ablation study on the critic and search hyper-parameters in HTPS.** We report the pass@8 score for Metamath, and the cumulative pass rate for Equations. Evariste, trained with a soft critic and stochastic hyper-parameters, obtains the best performance in both environments. Removing the critic, or using a hard critic leads to reduced performances. In Equations, adding stochasticity in the proof search hyper-parameters increases the performance by 4.3% in Equations, and slightly improves performance in Metamath.

	Evariste	No critic	Hard critic	Fixed search params
Metamath (valid)	68.6	64.8	67.6	**69.8**
Metamath (test)	**57.4**	52.2	57.4	56.2
Equations (*Identities*)	**78.1**	65.6	63.1	73.8

To measure the impact of our critic model, we run an experiment where the proof search is only guided by the policy model. In particular, during the back-propagation phase, we set $v_T(g) = 0.5$ for leaves of T. In that context, our model is no longer trained with a critic objective. We report the results in Table 5. We find that using a critic model improves the performance significantly, by 5.2% and 12.5% on Metamath and Equations respectively.

As mentioned in Section 4, to train the critic objective, we set the training targets as $c(g) = 1$ for solved nodes, $c(g) = 0$ for invalid nodes and $c(g) = W(g, t^*)/N(g, t^*)$ where t^* is the tactic that maximizes the search policy at g, for internal nodes. We also tested a hard critic estimation of the target values, following Polu and Sutskever [9], where $c(g) = 1$ for solved nodes and $c(g) = 0$ for both invalid and internal nodes. We report results in Table 5. For both Metamath and Equations, HTPS critic targets allow Evariste to reach its best performance. In Equations, the model reaches a cumulative pass rate of 78.1%, compared to 63.1% with hard critic estimates. In Equations, using hard critic targets gives worse performances than having no critic model at all, showing that these targets are a bad estimation: setting all internal nodes to zero is too pessimistic.

6.2.3 Fixed proof search parameters

We study the impact of sampling HTPS hyper-parameters for each attempt during online training. We run experiments with fixed, chosen search parameters for Equations and Metamath to compare with random sampling, and report results in Table 5. Evariste achieves better performances than the model trained with fixed search parameters on Metamath test set and Equations *Identities*, reaching 78.1% pass rate compared to 73.8% in Equations *Identities*.

6.2.4 Model update frequency during online training

In our online training procedure, the policy and critic models are updated every five minutes on the provers. We measure the impact of the frequency of these updates by trying different refresh rates: 5 minutes, 1 hour, 6 hours for Equations, and no updates at all for both Equations and Metamath. We report the cumulative pass rate over training hours in Figure 3. The higher the refresh rate, the better the cumulative pass rate over time, confirming the benefits of online training over expert iteration.

7 Conclusion

In this work, we introduce HTPS, an AlphaZero-inspired proof search algorithm for automated theorem proving, along with an online training procedure. We run an extensive study of our pipeline, and present state-of-the-art results on multiple proving environments. We show that online training provides large speed-ups over expert iteration, and allows generalization of the policy and critic models to completely new domains. Despite large number of attempts per theorem, proving the entirety of datasets like miniF2F remains elusive, and generating data from proof-search on the currently available corpora will likely be insufficient in the long term. As manually annotated formal datasets are limited, another way of providing exploration and additional training data (in the spirit of self-play for two player games) is required. Automated generation of new theorems is likely to be one of the future milestones.

References

[1] Alfred B Kempe. On the geographical problem of the four colours. *American journal of mathematics*, 2(3):193–200, 1879.

[2] Vladimir Voevodsky. Univalent foundations of mathematics. In *International Workshop on Logic, Language, Information, and Computation*, pages 4–4. Springer, 2011.

[3] Thomas Hales, Mark Adams, Gertrud Bauer, Dat Dang, John Harrison, Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Nguyen, Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Ta, Trân Trung, Diep Trieu, and Roland Zumkeller. A formal proof of the kepler conjecture. *Forum of Mathematics, Pi*, 5, 01 2017. doi: 10.1017/fmp.2017.1.

[4] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

[7] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM J. Res. Dev.*, 4(1):28–35, jan 1960. ISSN 0018-8646. doi: 10.1147/rd.41.0028. URL https://doi.org/10.1147/rd.41.0028.

[8] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7 (3):201–215, jul 1960. ISSN 0004-5411. doi: 10.1145/321033.321034. URL https://doi.org/10.1145/321033.321034.

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

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

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

[12] Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic. *arXiv preprint arXiv:1608.02644*, 2016.

[13] Bruce Abramson and Richard E Korf. A model of two-player evaluation functions. In *AAAI*, pages 90–94, 1987.

[14] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[15] Norman D. Megill and David A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. http://us.metamath.org/downloads/metamath.pdf.

[16] The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, jan 2020. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145%2F3372885.3373824.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[19] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[20] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[21] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, 2015.

[22] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1eZYeHFDS.

[23] Stephan Schulz. E—a brainiac theorem prover. *AI Commun.*, 15(2–3):111–126, 2002.

[24] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In *IJCAR*, 2001.

[25] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[26] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[28] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, pages 454–463. PMLR, 2019.

[29] John Harrison. Hol light: A tutorial introduction. In *International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

[30] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: learning to prove with tactics. *Journal of Automated Reasoning*, 65(2):257–286, 2021.

[31] Karel Chvalovskỳ, Jan Jakubův, Miroslav Olšák, and Josef Urban. Learning theorem proving components. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 266–278. Springer, 2021.

[32] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. *arXiv preprint arXiv:2006.03511*, 2020.

[33] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gR5iR5FX.

[34] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

[35] Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=m5Qsh0kBQG.

[36] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*, 2020.

[37] Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR, 2020.

[38] Guillaume MJ-B Chaslot, Mark HM Winands, and HJVD Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pages 60–71. Springer, 2008.

[39] Yizao Wang and Sylvain Gelly. Modifications of uct and sequence-like simulations for monte-carlo go. In *2007 IEEE Symposium on Computational Intelligence and Games*, pages 175–182. IEEE, 2007.

[40] Mark HM Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-carlo tree search solver. In *International Conference on Computers and Games*, pages 25–36. Springer, 2008.

[41] Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. *arXiv preprint arXiv:2006.10369*, 2020.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[43] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[44] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019.

[45] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[47] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SylO2yStDr`.

[48] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS 2017 Autodiff Workshop*, 2017.

8 Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 6.1 includes our results as well as their limitations.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Beyond environmental cost of training models, we do not see any direct path from the research presented in this paper to negative applications.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code for the Equations environment will be open-sourced. We also plan to make our trained model publicly available to help people in the formal community. Code for the overall distributed training architecture is tied to our infrastructure and will be difficult to open-source.

(b) Did you specify all the training details (e.g., data splits, hyper-parameters, how they were chosen)? [Yes] We provide all training details in Section D.2 of the Appendix.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] As stated in Section 5.2, we report the result of one training and one evaluation as running all evaluations multiple times would be too costly.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [No] License is available on the repositories associated with each asset.

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A] No new assets in this work.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]