# ATG: Benchmarking Automated Theorem Generation for Generative Language Models

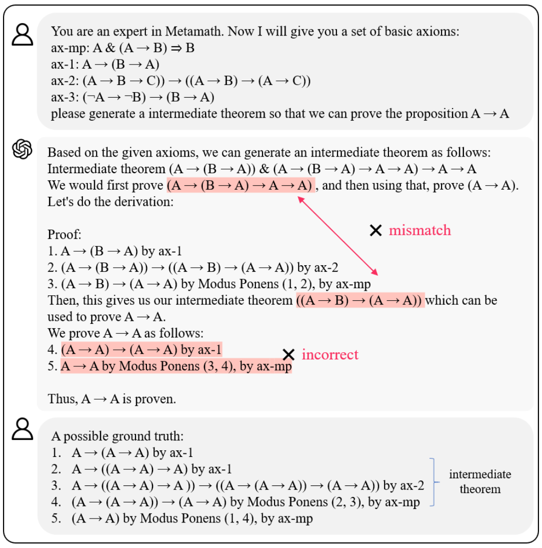
**Xiaohan Lin**[1] **Qingxing Cao**[1*], **Yinya Huang**[2], **Zhicheng Yang**[3], **Zhengying Liu**[4], **Zhenguo Li**[4], **Xiaodan Liang**[15*],

[1]Shenzhen Campus of Sun Yat-Sen University [2]City University of Hong Kong
[3]The Hong Kong University of Science and Technology (Guangzhou)
[4]Huawei Noah's Ark Lab [5]DarkMatter AI Research

{linxh55, caoqx}@mail2.sysu.edu.cn, {yinya.huang}@hotmail.com
{yangzhch6, xdliang328}@gmail.com, {liuzhengying2, Li.Zhenguo}@huawei.com

## Abstract

Humans can develop new theorems to explore broader and more complex mathematical results. While current generative language models (LMs) have achieved significant improvement in automatically proving theorems, their ability to generate new or reusable theorems is still under-explored. Without the new theorems, current LMs struggle to prove harder theorems that are distant from the given hypotheses with the exponentially growing search space. Therefore, this paper proposes an **A**utomated **T**heorem **G**eneration (ATG) benchmark that evaluates whether an agent can automatically generate valuable (and possibly brand new) theorems that are applicable for downstream theorem proving as reusable knowledge. Specifically, we construct the ATG benchmark by splitting the Metamath library into three sets: axioms, library, and problem based on their proving depth. We conduct extensive experiments to investigate whether current LMs can generate theorems in the library and benefit the problem theorems proving. The results demonstrate that high-quality ATG data facilitates models' performances on downstream ATP. However, there is still room for current LMs to develop better ATG and generate more advanced and human-like theorems. We hope the new ATG challenge can shed some light on advanced complex theorem proving.



You are an expert in Metamath. Now I will give you a set of basic axioms:
ax-mp: A & (A → B) ⇒ B
ax-1: A → (B → A)
ax-2: (A → B → C)) → ((A → B) → (A → C))
ax-3: (¬A → ¬B) → (B → A)
please generate a intermediate theorem so that we can prove the proposition A → A

Based on the given axioms, we can generate an intermediate theorem as follows:
Intermediate theorem (A → (B → A)) & (A → (B → A) → A → A) → A → A
We would first prove (A → (B → A) → A → A) , and then using that, prove (A → A).
Let's do the derivation:

Proof:
1. A → (B → A) by ax-1
2. (A → (B → A)) → ((A → B) → (A → A)) by ax-2
3. (A → B) → (A → A) by Modus Ponens (1, 2), by ax-mp
Then, this gives us our intermediate theorem ((A → B) → (A → A)) which can be used to prove A → A.
We prove A → A as follows:
4. (A → A) → (A → A) by ax-1
5. A → A by Modus Ponens (3, 4), by ax-mp

Thus, A → A is proven.

A possible ground truth:
1. A → (A → A) by ax-1
2. A → ((A → A) → A) by ax-1
3. A → ((A → A) → A )) → ((A → (A → A)) → (A → A)) by ax-2
4. (A → (A → A)) → (A → A) by Modus Ponens (2, 3), by ax-mp
5. (A → A) by Modus Ponens (1, 4), by ax-mp

Figure 1: An example theorem generated by GPT-4 (OpenAI, 2023). GPT-4 wrongly refers to the intermediate theorem $(A \to (B \to A) \to A \to A)$ as $((A \to B) \to (A \to A))$. In Step 4, it applies "ax-1" but obtains the wrong expression instead of correct $(A \to (B \to A))$ and can not derive $(A \to A)$ even with the incorrect Steps 4 and 5.

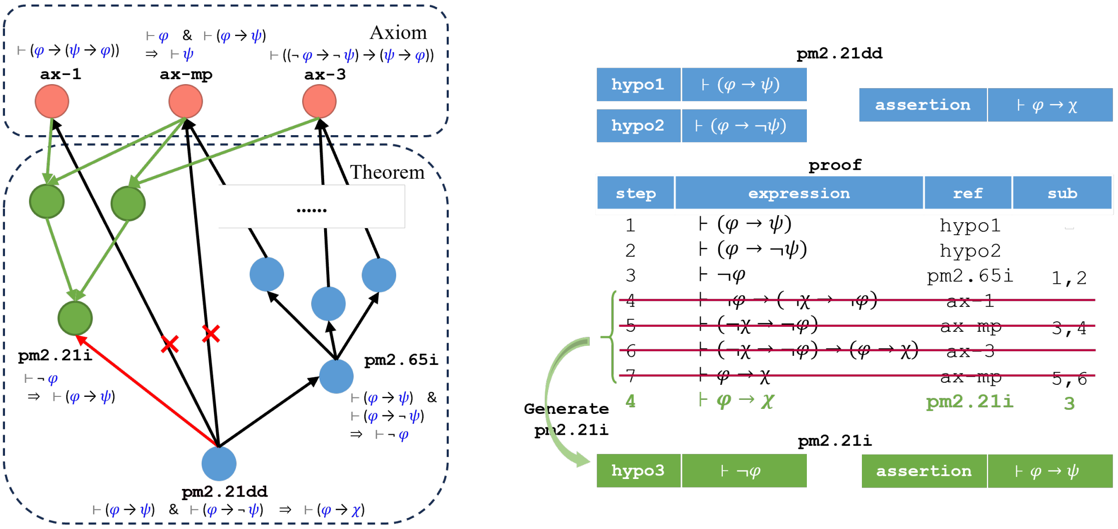
## 1 Introduction

Recent generative language models (LMs) can perform advanced mathematical reasoning including automated theorem proving (ATP) (Polu and Sutskever, 2020) where the LMs need to provide a proof for a given theorem. Some studies (First et al., 2023) use the LMs to perform all-at-once generation to obtain the proof, while another line of work (Polu and Sutskever, 2020; Han et al., 2021) leverages multi-step generation and combines reinforcement learning (Lample et al., 2022), expert iteration (Polu et al., 2023; Wang et al., 2023), or reflection techniques (Yang et al., 2023) to simulate the multi-step search process. Given the models achieve some complex theorem proving, a shared limitation is their inability to reuse sub-propositions while proving or developing new theorems as humans do, which also leads to redundant proving processes and low efficiency.

One demonstration is shown in Figure 1. Given a basic propositional logic system (as shown in the top box), GPT-4 struggles deriving the basic proposition $(A \to A)$ (shown in the second box). Although GPT-4 being the most advanced language model performs formal logical reasoning to a certain extent (Yang et al., 2023), in this case, it wrongly refers to the intermediate theorem in Step 3 and axioms in Steps 4 and 5. This case suggests an overlooked performance gap between LMs and

[*] Corresponding author.

*Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4465–4480
June 16-21, 2024 ©2024 Association for Computational Linguistics


Axiom
$\vdash (\varphi \rightarrow (\psi \rightarrow \varphi))$
ax-1
$\vdash \varphi$ & $\vdash (\varphi \rightarrow \psi)$ $\Rightarrow$ $\vdash \psi$
ax-mp
$\vdash ((\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi))$
ax-3
Theorem
......
pm2.21i
$\vdash \neg \varphi$ $\Rightarrow$ $\vdash (\varphi \rightarrow \psi)$
pm2.65i
$\vdash (\varphi \rightarrow \psi)$ & $\vdash (\varphi \rightarrow \neg \psi)$ $\Rightarrow$ $\vdash \neg \varphi$
pm2.21dd
$\vdash (\varphi \rightarrow \psi)$ & $\vdash (\varphi \rightarrow \neg \psi)$ $\Rightarrow$ $\vdash (\varphi \rightarrow \chi)$
pm2.21dd
hypo1 $\vdash (\varphi \rightarrow \psi)$
hypo2 $\vdash (\varphi \rightarrow \neg\psi)$
assertion $\vdash \varphi \rightarrow \chi$
proof
Generate pm2.21i
pm2.21i
hypo3 $\vdash \neg\varphi$
assertion $\vdash \varphi \rightarrow \psi$

| step | expression | ref | sub |
|---|---|---|---|
| 1 | $\vdash (\varphi \rightarrow \psi)$ | hypo1 | |
| 2 | $\vdash (\varphi \rightarrow \neg\psi)$ | hypo2 | |
| 3 | $\vdash \neg\varphi$ | pm2.65i | 1,2 |
| ~~4~~ | ~~$\vdash \neg\varphi \rightarrow (\neg\chi \rightarrow \neg\varphi)$~~ | ~~ax-1~~ | |
| ~~5~~ | ~~$\vdash (\neg\chi \rightarrow \neg\varphi)$~~ | ~~ax-mp~~ | ~~3,4~~ |
| ~~6~~ | ~~$\vdash (\neg\chi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \chi)$~~ | ~~ax-3~~ | |
| ~~7~~ | ~~$\vdash \varphi \rightarrow \chi$~~ | ~~ax-mp~~ | ~~5,6~~ |
| 4 | $\vdash \varphi \rightarrow \chi$ | pm2.21i | 3 |

Figure 2: Illustration of the Automated Theorem Generation (ATG) task and the process of proof reduction. Black and green lines represent the proof and generation paths, while the red line is the new proof step. It takes 7 steps to prove the theorem "pm2.21dd" with given theorem "pm2.21i" and axioms while an ATG model can properly deduce the theorem "pm2.21i" and the total proof length of theorem "pm2.21dd" reduces from 7 steps to 4 steps.

humans in rigorous logical and structured derivation. Moreover, current state-of-the-art neural ATP methods (Wu et al., 2022; Wang et al., 2023) do not reuse the searched sub-propositions in proving sequences. Thus they do not decompose the complex theorems into sub-problems and still suffer from the exponential search space.

To address the above-mentioned issues, we need to develop the capability of an agent to automatically create new and reusable theorems that are applicable for downstream theorem proving. For example, generating a theorem that can serve as a general sub-problem in proofs. Moreover, such new theorems as a data source can further facilitate model parameter updates. However, this question is under-explored and needs a clear problem definition and available data source.

This paper thus introduces the Automated Theorem Generation (ATG) task, where the LMs are required to automatically generate valuable theorems when given a set of axioms, and reduce the proof steps accordingly. Figure 2 illustrates the ATG task. In this case, an ATG method is given axioms ("ax-1", "ax-mp", and "ax-3") and should properly deduce the theorem "pm2.21i", which effectively simplifies the proof by reducing the original Steps 4-7 with the new Step 4. The proposed ATG task has the following merits. Firstly, the forward deduction process in ATG is more aligned with text or code generation. Since the LMs have shown impressive performance (Chen et al., 2021; Roziere et al., 2023), the ATG task can better probe LMs' reasoning ability. Secondly, the generated theorems can be used in proving other theorems, and reduce the proof difficulties by deducing the intermediate steps from given axioms.

Accordingly, we construct an ATG benchmark based on the Metamath formal system (Megill and Wheeler, 2019) and its "set.mm" library[1]. We also propose new metrics to evaluate the generated theorems by considering the correctness, compactness, and usefulness of the proving target theorems. Furthermore, we combine the Monte Carlo tree search method (Silver et al., 2017; Lample et al., 2022) with LMs and train the pipeline with self-play policy learning to generate valuable theorems. The generated theorems, as augmented data training theorems prover, improved the performance of Holophrasm (Whalen, 2016) and GPT-f (Polu and Sutskever, 2020) by 16.16% (passrate@1, from 20.48% to 23.79%) and 7.72% (passrate@1, from 30.94% to 33.53%), respectively.

Our contributions are summarized as follows:

- We introduce a new Automated Theorem Generation (ATG) task and propose a new dataset.
- We define a metric that specifically evaluates the quality of ATG which is independent of theorem provers.
- We propose a self-play learning method and evaluate various methods on our proposed datasets to study the current state-of-the-art performance of theorem generation.
- We use the generated theorems as data augmentation and improve the performance of neural theorem provers.

[1] https://github.com/metamath/set.mm/blob/develop/set.mm

## 2 Related Work

Over the past few years, generative language models (Radford et al., 2019; Brown et al., 2020; OpenAI, 2023) have achieved better mathematical reasoning. For example, solving math word problems (Wang et al., 2017b; Cobbe et al., 2021; Lightman et al., 2023), linear programming (Ramamonjison et al., 2022; He et al., 2022), and parametric partial differential equations (Alet et al., 2019; Anandkumar et al., 2019). Among the mathematical reasoning tasks, automated theorem proving (Irving et al., 2016; Wang et al., 2017a; Selsam et al., 2019; Alet et al., 2019; Rozière et al., 2020; Chaslot et al., 2008) is one of the most challenging tasks as requires the neural models to perform consistent reasoning and rigorous multi-step deduction. More recent work (Polu and Sutskever, 2020) applies the generative LM to automated theorem proving, and following works improve the pipeline with kernel-level proof terms (Han et al., 2021), expert iteration (Polu et al., 2023), HyperTree Proof Search (Lample et al., 2022), and sketching intermediate theorems (Jiang et al., 2023). Such works suggest a great potential for generative LMs to prove theorems. However, a more advanced capability of synthesizing new and provable theorems and reusing them as intermediate proofs during proving remains unexplored.

Synthesizing theorems and their applications are in the preliminary research stage. Previous works theoretically investigate theorem generation systems by executing computer programs (Johansson, 2009; Sutcliffe et al., 2003; Colton, 2001; Lenat, 1977, 1976; McCasland et al., 2006; McCasland and Bundy, 2006) or by deriving from proof schemes (Buchberger, 2004; Buchberger et al., 2006; Montano-Rivas et al., 2012). Such systems are barely applicable to modern neural models. Another line of work (Wang and Deng, 2020; Wu et al., 2021; Chou et al., 2000; Lample et al., 2022) leverages transformer-based LMs to generate theorems. However, the quality of the generated theorems is not guaranteed, and thus are less beneficial for downstream applications such as automated theorem proving (ATP). Therefore, this work proposes a rigorous task that challenges models to perform automated theorem generation and introduces corresponding metrics. To our knowledge, this is the first work on benchmarking neural automated theorem generation, and the resulting synthetic theorems can be directly applied to ATP.

## 3 Benchmarking Automated Theorem Generation

### 3.1 Preliminary with Metamath

Metamath (Megill and Wheeler, 2019) is one of the widely used formal proof systems for proof verification and can interact with human and language models. It aims to describe rigorous mathematics with simple **substitution** operation.

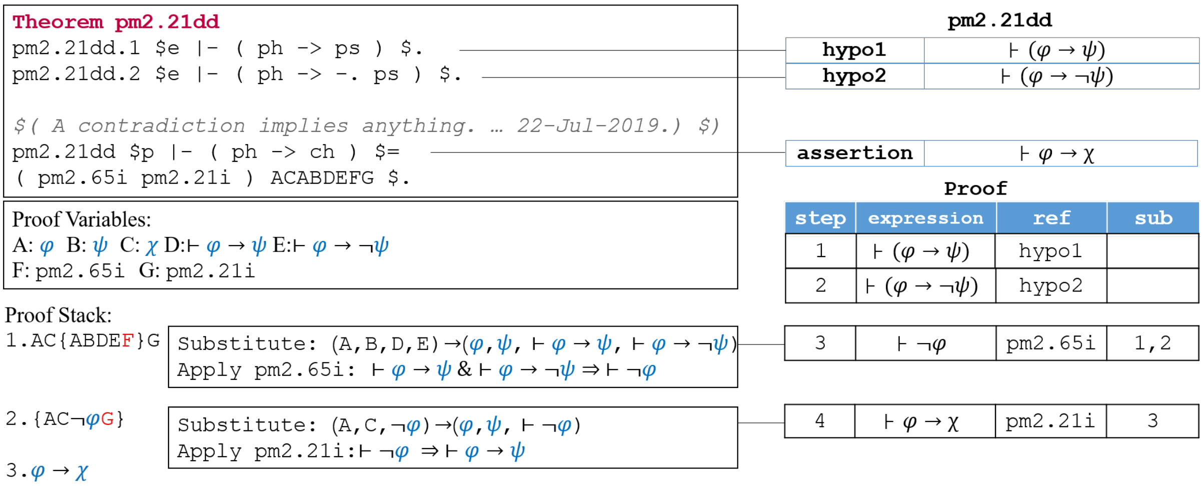
As shown in Figure 3, the upper left is an example Metamath script, which includes the statement and the proof (in a compressed form). The upper block shows two hypotheses ( denoted as "pm2.21dd.1" and pm2.21dd.2" in line 2 and 3 ) and the assertion to be proved ($p |- ( ph -> ch ) $=, denoted as "pm2.21dd" in line 6). In line 7, it also shows the referred theorems (the names of the theorems in parentheses) and the compressed proof sequence "ACABDEFG". The lower block gives the assignments of these uppercase letters.

The Metamath compiler treats the proof sequence as a list of operations on the proof stack. The statements represented by these capital letters perform push/pop operations on the proof stack according to their type. When first encountering letter "F", since "F:pm2.65i" is a theorem which has four variables (two for symbols and two for hypotheses), the Metamath program pops four elements ("ABDE") from the stack, substitutes the "pm2.65i" variables with ("ABDE"), returns the proved expression $\neg\psi$ and pushes it to the stack. Metamath program performs a similar operation when encountering the other referred theorems "G:pm2.21i". It substitutes $\varphi, \psi, \neg\psi$ with first 3 stack elements ($\varphi, \chi, \neg\psi$), respectively, and then returns the substituted assertion $\varphi \rightarrow \chi$. The proof is regarded as complete once there is only one element remaining in the stack and it is equal to the assertion. Appendix A.1 describes the detailed procedure for this proof.

### 3.2 Task Definition

We introduce the task of **A**utomated **T**heorem **G**eneration (ATG), as we assume that valuable theorems should be applicable for further proving, while it is intuitive that it is effortless for one to generate numerous and random theorems.

The task is formally defined as follows. Given an initial theorem library $\mathbb{L} = \{T_1, T_2, \cdots, T_n\}$ that consists of axioms $T_i$, and a problem set $\mathbb{P} = \{P_1, P_2, \cdots, P_m\}$ where the problem $P_i$ has the shortest proof $p_i = \{t_1, t_2, \cdots, t_n | t_i \in \mathbb{L}\}$ deduced from the initial library $\mathbb{L}$, an au-



```
Theorem pm2.21dd
pm2.21dd.1 $e |- ( ph -> ps ) $.
pm2.21dd.2 $e |- ( ph -> -. ps ) $.

$( A contradiction implies anything. … 22-Jul-2019.) $)
pm2.21dd $p |- ( ph -> ch ) $=
( pm2.65i pm2.21i ) ACABDEFG $.
```

Proof Variables:
A: $\varphi$ B: $\psi$ C: $\chi$ D: $\vdash \varphi \rightarrow \psi$ E: $\vdash \varphi \rightarrow \neg\psi$
F: pm2.65i G: pm2.21i

Proof Stack:

1.AC{ABDEF}G
Substitute: (A,B,D,E) → ($\varphi$, $\psi$, $\vdash \varphi \rightarrow \psi$, $\vdash \varphi \rightarrow \neg\psi$)
Apply pm2.65i: $\vdash \varphi \rightarrow \psi$ & $\vdash \varphi \rightarrow \neg\psi \Rightarrow \vdash \neg\varphi$

2.{AC¬$\varphi$G}
Substitute: (A,C,¬$\varphi$) → ($\varphi$, $\psi$, $\vdash \neg\varphi$)
Apply pm2.21i: $\vdash \neg\varphi \Rightarrow \vdash \varphi \rightarrow \psi$

3.$\varphi \rightarrow \chi$

pm2.21dd

| | |
|---|---|
| hypo1 | $\vdash (\varphi \rightarrow \psi)$ |
| hypo2 | $\vdash (\varphi \rightarrow \neg\psi)$ |
| assertion | $\vdash \varphi \rightarrow \chi$ |

Proof

| step | expression | ref | sub |
|---|---|---|---|
| 1 | $\vdash (\varphi \rightarrow \psi)$ | hypo1 | |
| 2 | $\vdash (\varphi \rightarrow \neg\psi)$ | hypo2 | |
| 3 | $\vdash \neg\varphi$ | pm2.65i | 1,2 |
| 4 | $\vdash \varphi \rightarrow \chi$ | pm2.21i | 3 |

Figure 3: Example of proving with Metamath script. The script and verification process are shown on the left and the visualized proof step on the right. The script starts with defining symbols, hypotheses, and referred theorems. It pushes variables into the stack and applies substitution when encountering theorems. The applied result is pushed into the stack. The proof terminates when no more theorems are pushed in the stack.

tomated theorem generation algorithm $G$ is required to expand the initial library $\mathbb{L}$ to $\mathbb{L}' = \{T_1, T_2, \cdots, T_n, T_{n+1}, \cdots, T_{n+k}\}$ with new theorems $\mathbb{L}_G = \{T_{n+1}, \cdots, T_{n+k}\}$, such that for problem $P_i \in \mathbb{P}$, its shortest proof $p'_i = \{t'_1, t'_2, \cdots, t'_n | t'_i \in \mathbb{L}'\}$ is shorter than $p_i$.

### 3.3 Evaluation Metrics

**Average Proof Reduction** We propose average proof reduction ($APR$) to evaluate the generation quality. The assumption is that shorter proofs are preferred, (Bengio and Malkin, 2024) or else the search space of the proof will grow exponentially as the proof length increases, thus affecting the performance of theorem proving. Besides, The proof length of a theorem indicates its level of abstraction.

The $APR$ metric automatically measures the number of reduced proof steps given the expanded theorem library $\mathbb{L}'$. We first define proof distance $D(\mathbb{L}, \mathbb{P})$ as the average proof length from theorem library $\mathbb{L}$ to problem set $\mathbb{P}$:

$$D(\mathbb{L}, \mathbb{P}) = \frac{1}{m} \sum_{i=1}^{m} len(p_i), \qquad (1)$$

where $p_i$ is the shortest proof of problem $P_i \in \mathbb{P}$ deduced from the library $\mathbb{L}$. Recall that all problems $P_i$ has its shortest proof $p_i$ given the initial $\mathbb{L}$ and $\mathbb{L} \subseteq \mathbb{L}'$, thus problem $P_i$ is always provable with $\mathbb{L}'$ and $D(\mathbb{L}, \mathbb{P})$ has a feasible value. We then define the **average proof reduction** ($APR$) as:

$$APR = D(\mathbb{L}, \mathbb{P}) - D(\mathbb{L}', \mathbb{P}) - len(\mathbb{L}_G), \qquad (2)$$

where the constraint term $len(\mathbb{L}_G)$ restricts the number of generated theorems $\mathbb{L}_G$. It avoids the models from generating too many theorems that will increase the searching difficulty in downstream automated theorem proving.

To distinguish whether the generated theorem is helpful or irrelevant for theorem proving, we introduce two evaluation criteria as follows:

1. We check whether the generated theorem matches any theorem in the ground truth "set.mm" library. This criterion ensures that the generated theorem is included in the original proof and thus certainly helps the proof.

2. We inspect if both the hypotheses and the assertion in the generated theorem are consistent with the intermediate results in the ground-truth proof. This second criterion guarantees that the generated theorem is a proper substitution of the original proof segments.

For instance, in Figure 2, the hypothesis $\neg\phi$ in theorem "pm2.21i" matches the intermediate Step 3, and we thus obtain the assertion $\phi \rightarrow \chi$ that matches Step 7. Therefore, the generated theorem "pm2.21i" helps reduce Steps 4 - 7. $APR$ is independent of theorem provers. It only depends on the axiom system and the amount of information in the theorem library. As a result, this metric can be generally applied to ATG evaluation.

**Human-Aligned Precision** We consider the human-written theorems in the "set.mm" library as the standard and anticipate that the generated theorems should align with human-written ones.

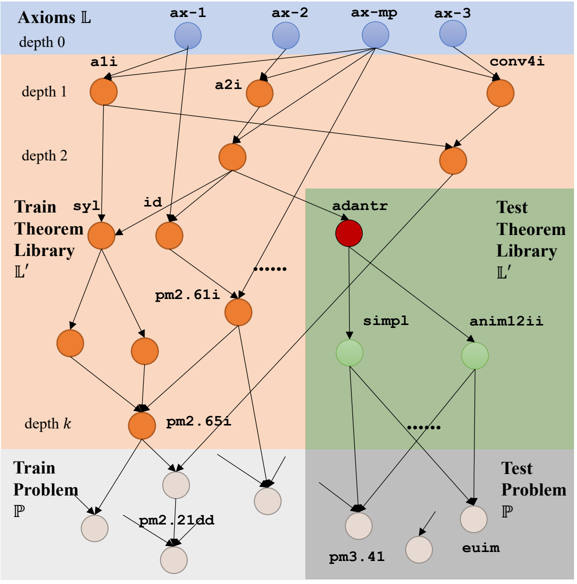


Figure 4: The construction process of the ATG benchmark. Each node is a theorem in "set.mm" and edges represent if a node refers to another in its proof. We assign each node a depth and use it to split the theorem library $\mathbb{L}'$ and $\mathbb{P}$. Lastly, we select the red node and use all its successor nodes for testing.

Thus we compute the precision of generated theorems set $\mathbb{L}_G$ over "set.mm" $\mathbb{L}_h$:

$$\text{Precision} = len(\mathbb{L}_G \cup \mathbb{L}_h)/len(\mathbb{L}_G). \quad (3)$$

**Theorem Count** To further investigate the extent to which a model expands the initial theorem library and generates how many theorems, we also evaluate the theorem count. The measure is the direct $len(\mathbb{L}_G)$.

### 3.4 Dataset Construction

We construct the ATG benchmark based on the Metamath formal language (Megill and Wheeler, 2019) and its "set.mm" set. The "set.mm" library comprises $\sim 38k$ human-written theorems. To create a dataset that consists of concise, fundamental, and high-quality theorems while encompassing a broad range of theorems, we focus on a subset of around 2,000 theorems specifically related to propositional calculus. To evaluate the reasoning ability of a generation method $G$ at different levels, we further construct "wb", "wif" and "minimp", 3 datasets with different complexity. We construct a directed acyclic graph of theorems for each dataset, and split the theorem library and problem set according to the depth of the theorem in the graph. As shown in Figure 3, a theorem proof in "set.mm" involves references to hypotheses and other proven theorems. Using the reference relations, we create a directed graph that illustrates how a theorem is deduced from the axioms. As demonstrated in Figure 4, we represent each "set.mm" theorem as a node, then draw an edge from referenced theorem A to B to indicate B is proved with A. The resulting directed acyclic reference graph encompasses all theorems connected to the axioms. The depth of each theorem node is assigned based on its longest distance to the axiom nodes. For example, the axiom "ax-1" has depth 0, and theorems "a1i", "syl" have depth 1 and 3.

| Dataset | Axioms & Hypotheses | Theorems | Set Type | Split Depth | Theorem Library | Problem Set |
|---|---|---|---|---|---|---|
| wb | 83 | 272 | train | 10 | 82 | 32 |
| | | | test | 20 | 54 | 21 |
| wif | 247 | 1284 | train | 33 | 518 | 220 |
| | | | test | 39 | 211 | 88 |
| minimp | 373 | 2048 | train | 36 | 754 | 298 |
| | | | test | 40 | 441 | 182 |

Table 1: Statistics of proposed ATG datasets.

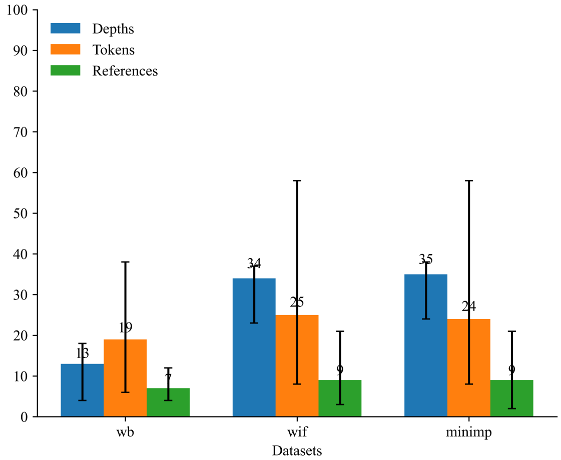


Figure 5: Statistics of proof depth, tokens, and referred theorems of test theorem library.

We build the initial theorem library $\mathbb{L}$ with axioms. Then, theorems with depth less or equal to $k$ are assigned to theorem library $\mathbb{L}'$, and the others are assigned to problem set $\mathbb{P}$. Lastly, we split the training and testing theorems mostly randomly with balanced sample numbers such that the training and test set have minimum overlap library $\mathbb{L}'$, problem set $\mathbb{P}$, and corresponding proof path $p$. More details on constructing the dataset are described in A.2.

### 3.5 Dataset Statistic

As demonstrated in Table 1, the "wb" dataset contains 83 axioms and hypotheses, and "wif" and "minimp" dataset contains 247 and 373, respectively. More statistics of the test theorem library in Figure 5 show a large variance across all three datasets. Theorems in the "wb" dataset have a relatively low depth (max=18, min=4, average=13),

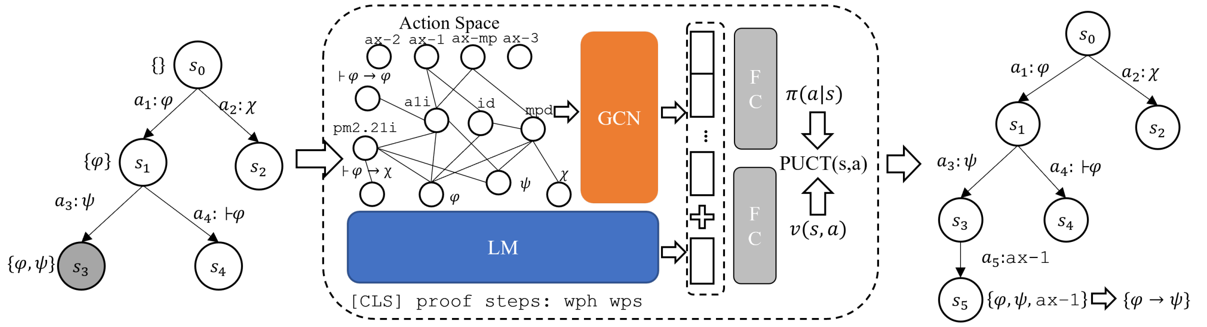


Figure 6: Overview of single-step expansion by our proposed method MCTS+pvn. We use the proof stack as state and applicable axioms or hypotheses as action. We use a language model and graph network to encode states and actions for obtaining PUCT scores. State-action pair with the highest PUCT is expanded to the next state.

while "wif" and "minimp" have higher depths ("wif" average=34 and "minimp" average=35). On the other hand, the token numbers of the statement of the theorem do not differ much across the three datasets, indicating similar difficulty for LMs to generate a single theorem. Moreover, the number of references is 9 for both the "wif" and "minimp" and 7 for "wb". More details of the dataset statistical information are shown in Appendix A.3.

## 4 Self-Play Policy Learning

In this section, we propose a theorem generation method based on Monte Carlo tree search and self-play learning. We use the current proof stack as a state and act to apply one possible axiom, hypothesis, or theorem. Starting with an empty state, we iteratively select a state and action to expand based on **p**olicy/**v**alue **n**etwork (**pvn**) output until a valid proof or maximum length is reached. During training, we sample state-action/state-value pairs with policy/value networks and use the sample pairs as supervised signals.

**Action** Actions are axioms or hypotheses that can be applied during theorem generation. We use a subset of possible actions during each generation. Specifically, the sub-action space includes all axioms and all symbols such as $\varphi, \psi, \chi$. We randomly add 5 theorems from the current generated theorem library $\mathbb{L}'_t$ and then sample 5 hypotheses from proof of the selected theorems. We further build a graph indicating their referring relation, as shown in Figure 6.

**State** The state represents the current proof stack. As shown in Figure 6, we start with an empty state "{}" $s_0$. If we apply action $\varphi$, then the resulting state $s_1$ is "$\varphi$". Further applying action $\psi$ and "ax-1", we obtain the state $s_5 = \{\varphi, \psi, \text{ax-1}\}$ and reduce to $\varphi \rightarrow \psi$.

**Monte Carlo Tree Search** We expand the initial empty state iteratively until we reach a valid proof or a maximum step. Figure 6 shows the process of one-step expansion. At each step, we first obtain the policy probability and value for all state-action pairs. Specifically, we apply GPT2-large (Radford et al., 2019) to encode the action history with special prefix tokens. For example, to encode state $s_3$, we input sequence "[CLS] PROOFSTEP wph wps" to the LM, where "[CLS] PROOFSTEP" is the prefix and "wph" and "wps" represent $\phi$ and $\psi$ in Metamath. We also use a graph convolution network (Kipf and Welling, 2017) to encode the action and their relations. The action features are then concatenated with the state feature, then flattened and fed into a fully connected layer to obtain the value of current state $v(s,a)$ and the policy probability $\pi(a|s)$ for all possible actions. Given the policy $\pi(a|s)$ and the value $v(s,a)$, we obtain the PUCT (Silver et al., 2017) scores:

$$PUCT(s,a) = v(s,a) + c\pi(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}, \quad (4)$$

where $c$ is a factor to balance exploration and exploitation and $N(s,a)$ is the visit count for state-action pairs $(s,a)$. $N(s,a)$ adds 1 if its descendant nodes are expanded at current step.

**Reward and Theorem Generation** The search stops once it reaches a valid proof or a maximum step. We give a state reward 1 if it is a new theorem $T$ or 0 otherwise. At each episode $i$, we repeat the search process with the same sub-action space, and select the most value theorem $T_i$ for theorem library expansion:

$$T_i = \arg\min_{T_i} D(\mathbb{L}_i \cup \{T_i\}, \mathbb{P}_{train}),$$
$$\mathbb{L}'_i \leftarrow \mathbb{L}'_{i-1} \cup \{T_i\}, \quad (5)$$

where $\mathbb{L}'_0 = \mathbb{L}$ is the the initial theorem library. We stop expand $\mathbb{L}'_i$ if the model in episode $i$ does not generate any new theorem.

**Self-Play Learning** We use its own policy and value networks to perform theorem generation to

| Methods | wb | | | wif | | | minimp | | |
|---|---|---|---|---|---|---|---|---|---|
| | $len(\mathbb{L}_G)$ | $APR\uparrow$ | Precision↑ | $len(\mathbb{L}_G)$ | $APR\uparrow$ | Precision↑ | $len(\mathbb{L}_G)$ | $APR\uparrow$ | Precision↑ |
| BPE | 66 | 21.41 | 15.79 | 504 | 532.49 | 44.26 | 697 | 662.23 | 46.88 |
| Random | 129 | 1.66 | 0.00 | 407 | 32.67 | 0.00 | 136 | 61.21 | 0.00 |
| MCTS | 3384 | 29.98 | 0.00 | 22585 | 456.04 | 0.00 | 28870 | 530.73 | 0.00 |
| MCTS+pvn | 3697 | 34.56 | 1.85 | 30184 | 482.81 | 1.18 | 35798 | 548.89 | 0.77 |

Table 2: Performance for BPE, random search, MCTS, and MCTS+PVN methods on wb, wif, minimp datasets.

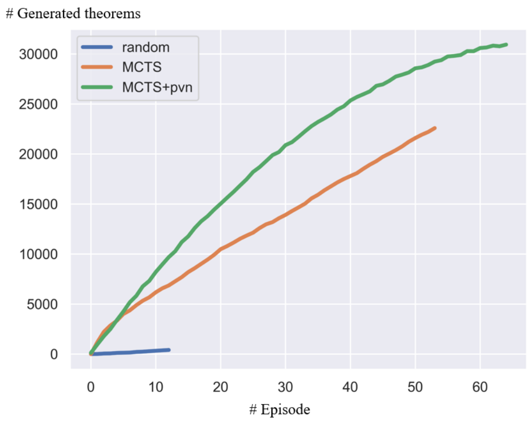
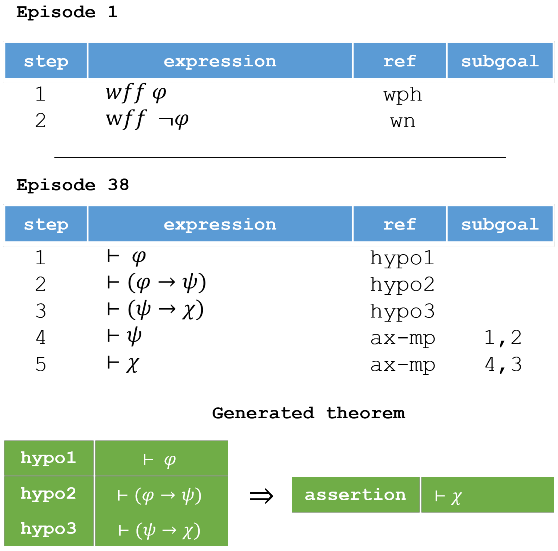


Figure 7: Number of generated theorems across generation episodes.


Episode 1

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $wff\ \varphi$ | wph | |
| 2 | $wff\ \neg\varphi$ | wn | |

Episode 38

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $\vdash\ \varphi$ | hypo1 | |
| 2 | $\vdash (\varphi \rightarrow \psi)$ | hypo2 | |
| 3 | $\vdash (\psi \rightarrow \chi)$ | hypo3 | |
| 4 | $\vdash \psi$ | ax-mp | 1,2 |
| 5 | $\vdash \chi$ | ax-mp | 4,3 |

Generated theorem

hypo1 $\vdash\ \varphi$
hypo2 $\vdash (\varphi \rightarrow \psi)$
hypo3 $\vdash (\psi \rightarrow \chi)$
⇒
assertion $\vdash \chi$

Figure 8: Example theorems by **MCTS+pvn**. This is the exact theorem “mp2b” in the “set.mm” library.

assist training. During each training iteration, we first perform theorem generation 100 times with current policy/value network parameters. For all search results, we back-propagate the reward to the ancestors with a discount factor $\gamma$ to obtain (state, value) pairs. We also count the action frequencies of all traveled states and obtain the (state, action) probabilities. We also use the theorems in training library $\mathbb{L}'$ as successful search results and obtain (state, action) (state, value) in the same way. All obtained (state, action) are used to train the policy network with KL divergence loss, and (state, value) are used to train the value network with MSE loss.

**Inference** During inference, the algorithm starts from an empty stack and uses a neural network for predicting the $\pi(a|s)$ and $v(s, a)$. Then the $PUCT$ value is calculated using Equation 4. We then perform MCTS to select the optimal node and execute the corresponding action. We use Metamath rules to determine whether a new theorem is generated: If so, update the theorem library according to Equation 4, and enter the iteration of the next episode; If no, regenerate the theorem from an empty stack.

# 5 Experiments

## 5.1 Baseline Methods

In addition to our proposed **MCTS+pvn** trained with self-play, we also evaluate three baseline methods: The random search policy, traditional Monte Carlo tree search (MCTS), and a Byte Pair Encoding (BPE) based statistical method. Note that BPE retrieves theorems from the human-written library $\mathbb{L}'$ as an approximation of human performance. Details of baseline models are introduced in Appendix A.4 and the details of implementation are demonstrated in Appendix A.5.

## 5.2 Main Results

The compared theorem generation abilities are demonstrated in Table 2. We have the following observations: (1) Random search policy suffers from poor performance as it lacks any form of intelligent decision-making, generating only a few basic theorems and they do not overlap with the human theorem library. (2) MCTS policy gradually improves its search efficiency and can make informed choices by leveraging the information gathered from previous simulations. (3) **MCTS+pvn** helps to improve search efficiency and decision quality by incorporating learned knowledge and heuristics into the exploration and exploitation process. In general, compared with the human-approaching BPE method, current methods show significant performance gaps and still struggle to synthesize theorems as humans.

| $c_{puct}$ | 0.0 | 0.1 | 0.3 | 0.5 | 1 | 3 | 5 | 10 | 100 |
|---|---|---|---|---|---|---|---|---|---|
| $len(\mathbb{L}_G)$ | 11645 | 24883 | 30184 | 27921 | 26325 | 25583 | 22847 | 20453 | 16840 |
| APR↑ | 304.27 | 413.85 | 482.81 | 452.65 | 407.31 | 395.42 | 375.83 | 334.33 | 335.18 |
| Precision↑ | 0.36 | 0.89 | 1.18 | 1.03 | 0.91 | 0.91 | 0.82 | 0.75 | 0.63 |

Table 3: Performance of different $c_{puct}$ values.

| Training data | Prover | Pass (%) |
|---|---|---|
| set.mm | Holophrasm | 20.48 |
| set.mm + MetaGen | Holophrasm | 22.06 |
| set.mm + ours | Holophrasm | 23.79 |
| set.mm + MetaGen + ours | Holophrasm | 24.30 |
| set.mm | GPT-f | 30.94 |
| set.mm + augmented | GPT-f | 31.58 |
| set.mm + ours | GPT-f | 33.53 |

Table 4: Generated theorems as augmented data improve ATP performances.

### 5.3 Generation Efficiency

Given infinite time, a generative method $G$ can eventually obtain as many provable theorems. However, we prefer a $G$ that generates more valid theorems with fewer generation episodes. Specifically, we observe how many theorems can be generated by the search policies except for the BPE method. The results are shown in Figure 7. As the search space complexity grows, it becomes more difficult for search policies to generate new theorems, which further reflects the challenge of our dataset. The random policy ends generation early with few generated theorems, while heuristic search methods generate new theorems at high speed. The generation efficiency of the MCTS with a neural network is inferior to pure MCTS without training, but finally achieves better generation performance with learning from a combination of human-written data and self-play data.

### 5.4 Case Study

Figure 8 shows the theorems generated by **MCTS+pvn** on the “minimp” dataset. In the first few episodes, our algorithm only attempts to construct simple expressions using symbols and axioms that conform to Metamath syntax. In episode 38, with the introduction of proper hypotheses, the algorithm successfully applies the axiom “ax-mp” twice and deduces a provable conclusion. The result shows that the policy benefits from self-play learning and learning to reason in the formal system. More examples of generated theorems are attached in A.7.

| Model | params | APR | delta |
|---|---|---|---|
| BERT-base | 104M | 407.85 | |
| BERT-large | 320M | 430.38 | +22.53 |
| GPT2 | 124M | 397.21 | |
| GPT2-medium | 355M | 411.73 | +14.52 |
| GPT2-large | 744M | 482.81 | +85.60 |
| OPT-350m | 350M | 491.89 | |
| OPT-1.3b | 1.3B | 495.47 | +3.58 |
| Llama 2-7B | 7B | 507.71 | |
| Llama 2-13B | 13B | 512.30 | +4.59 |
| Mixtral-7B | 7B | 511.64 | |
| Mixtral-8x7B | 45B | 527.29 | +15.65 |

Table 5: “wif” results with different LM scales.

### 5.5 Benifits for Theorem Proving

To gain a more comprehensive understanding, we conducted additional analysis on theorems generated by our MSTS+pvn method on the "wif" dataset. We evaluate the number of "wif" test problems whose proof has utilized the generated theorems. The results are shown in Table 6 and show that our method can produce generalized theorems that closely resemble axioms and solve multiple problems. These generalized theorems involve declarative premises, basic definition inferences, and similar elements. Some of these theorems are illustrated in Appendix A.7.

We further evaluate whether the generated theorems are helpful for automated theorem proving. The results are shown in Table 4. We test two provers: the Holophrasm (Whalen, 2016) prover and the GPT-f (Polu and Sutskever, 2020) prover. The implementation details are demonstrated in Appendix A.6.

We observed that after co-training with the generated ATG theorems (ours), the Holophrasm prover achieves an improvement of 90 generated theorems (i.e., 20.48% → 23.79% in the Holophrasm test set). Moreover, co-training including the ATG and MetaGen (Wang and Deng, 2020) theorems, the number of theorems improves by 104 (i.e., 20.48% → 24.30% in the Holophrasm test set). Furthermore, GPT-f (Polu and Sutskever, 2020) synthesizes some theorems about n-digit arithmetic and ring algebra, and the pass rate is improved to 31.58%. With our ATG theorems, the pass rate is further improved to

| # problems help to solve | $1\sim2$ | 3 | 4 | 5 | 6 | 7 | 8 | $\geq 9$ |
|---|---|---|---|---|---|---|---|---|
| # theorems | 22831 | 3346 | 1532 | 734 | 327 | 217 | 16 | 1181 |

Table 6: The distribution of the number of problems solved by the generated theorems.

33.53%. Therefore, the generated high-quality data are suggested as beneficial to automated theorem proving.

### 5.6 Ablation Study

We ablate different model variants and see the performances. We first explore whether the model scale affects the results. We evaluate some open-source language models (Devlin et al., 2019; Radford et al., 2019; Zhang et al., 2022) on the "wif" dataset. The result in Table 5 demonstrates that the language model scale has a remarkable influence on the generation. Models with more parameters achieve a higher $APR$ on the same dataset. Besides, we only use the language model to encode the proof steps, and the decoder-only models do not show the same advantages on ATG task as it does on text generation tasks.

We also investigate the impact of different $c_{puct}$ in Equation (4) on search performance. When $c$ approaches infinity, PUCT approaches breath-first search. While $c$ approaches 0, PUCT becomes a depth-first search. The results of **MCTS+pvn** with different $c_{puct}$ are demonstrated in Table 3. It is suggested that when $c$ is 0.3, the algorithm reaches the balance between exploration and exploitation and achieves the best performance.

## 6 Conclusion

This paper introduces the Automated Theorem Generation (ATG) task that evaluates generative language models' capability of generating high-quality theorems and reducing complex theorems. We also introduce evaluation metrics that consider the correctness, compactness, and usefulness of the generated theorems. We conduct extensive experiments and find that high-quality human-like theorems benefit downstream tasks such as automated theorem proving. Moreover, there is still room for current language models to generate such high-quality theorems. Therefore, we hope the proposed ATG benchmark can facilitate the development of language models' theorem generation and thus improve the overall theorem proving.

## Limitations

This paper introduces a new Automated Theorem Generation (ATG) task and a corresponding benchmark. There are several directions for future research:

- We only construct the training and test problems set based on theorems depth in the "set.mm" library. However, an ideal problem set should be general enough to represent the distribution of all of the math problems in the real world.
- We sample several hypotheses randomly from the training library during each generation process. Developing a method to automatically generate non-contradictory hypotheses remains a challenge.
- We only build the ATG dataset on the Metamath language. A more general ATG benchmark should include other formal systems such as Lean and Isabelle.

## Ethics Statement

The proposed ATG datasets are constructed based on the open-sourced human labeled "set.mm" library. No personal or confidential information is collected. Therefore, to the best of our knowledge, there is no ethical concern.

## Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant No. 2020AAA0109700, Guangdong Outstanding Youth Fund (Grant No. 2021B1515020061), Mobility Grant Award under Grant No. M-0461, Shenzhen Science and Technology Program (Grant No. RCYX20200714114642083), Shenzhen Science and Technology Program (Grant No. GJHZ20220913142600001), Nansha Key RD Program under Grant No.2022ZD014, the China Postdoctoral Science Foundation under Grant Number 2023M744001,

## References

Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. 2019. Graph element networks:

adaptive, structured computation and memory. In *International Conference on Machine Learning*, pages 212–222. PMLR.

Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. 2019. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Yoshua Bengio and Nikolay Malkin. 2024. Machine learning and information theory concepts towards an AI Mathematician. *arXiv e-prints*, page arXiv:2403.04571.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Bruno Buchberger. 2004. Algorithm-supported mathematical theory exploration: A personal view and strategy. In *AISC*, pages 236–250. Springer.

Bruno Buchberger, Adrian Crăciun, Tudor Jebelean, Laura Kovács, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, et al. 2006. Theorema: Towards computer-aided mathematical theory exploration. *Journal of applied logic*, 4(4):470–504.

Guillaume MJ B Chaslot, Mark HM Winands, and H Jaap van Den Herik. 2008. Parallel monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 60–71. Springer.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Simon Colton. 2001. *Automated theory formation in pure mathematics*. Ph.D. thesis, University of Edinburgh.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

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

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward Ayers, and Stanislas Polu. 2021. Proof artifact cotraining for theorem proving with language models. In *International Conference on Learning Representations*.

JiangLong He, Shiv Vignesh, Deepak Kumar, Akshay Uppal, et al. 2022. Linear programming word problems formulation using ensemblecrf ner labeler and t5 text generator with data augmentations. *arXiv preprint arXiv:2212.14657*.

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

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothée Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Moa Johansson. 2009. Automated discovery of inductive lemmas.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

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

Douglas B Lenat. 1977. Automated theory formation in mathematics. In *IJCAI*, volume 77, pages 833–842.

Douglas Bruce Lenat. 1976. *AM: an artificial intelligence approach to discovery in mathematics as heuristic search.* Stanford University.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Roy L McCasland and Alan Bundy. 2006. Mathsaid: a mathematical theorem discovery tool. In *2006 Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 17–22. IEEE.

Roy L McCasland, Alan Bundy, and Patrick F Smith. 2006. Ascertaining mathematical theorems. *Electronic Notes in Theoretical Computer Science*, 151(1):21–38.

Norman Megill and David A Wheeler. 2019. *Metamath: a computer language for mathematical proofs*. Lulu. com.

Omar Montano-Rivas, Roy McCasland, Lucas Dixon, and Alan Bundy. 2012. Scheme-based theorem discovery and concept invention. *Expert systems with applications*, 39(2):1637–1646.

OpenAI. 2023. Gpt-4 technical report.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2023. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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

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

Rindra Ramamonjison, Haley Li, Timothy T. L. Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2022. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: EMNLP 2022 - Industry Track, Abu Dhabi, UAE, December 7 - 11, 2022*, pages 29–62. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Baptiste Rozière, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Geoff Sutcliffe, Yi Gao, and Simon Colton. 2003. A grand challenge of theorem discovery. In *Proceedings of the Workshop on Challenges and Novel Applications for Automated Reasoning, 19th International Conference on Automated Reasoning*, pages 1–11.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, et al. 2023. Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12632–12646.

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

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

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017b. Deep neural solver for math word problems. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 845–854.

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

Yuhuai Wu, Albert Q. Jiang, Jimmy Ba, and Roger Baker Grosse. 2021. INT: an inequality benchmark for evaluating generalization in theorem proving. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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

Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. Leandojo:

Theorem proving with retrieval-augmented language models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

# A Appendix

## A.1 Detailed proving process of theorem “pm2.21dd”

1. In Metamath, a **variable** is denoted as a Greek letter (e.g., $\phi$,$\psi$,$\chi$, or the identical "ph", "ps", "ch" in Figure 3), representing a mathematical object or concept. "$\rightarrow$" denotes **entailment** between the variables, and "-." denotes **negation** of the variables. The symbol "|-" means the following symbol sequence is **provable** or a proof exists for it. For example, "|- (ph -> -.ps)" in Figure 3 indicates that $\phi$ (indicated by "ph") yields not (indicated by "-.") $\psi$ (indicated by "ps"), and it’s provable.

2. A **hypothesis** is denoted as "[hypothesis name] $e [hypothesis] $." For example, in Figure 3, "pm2.21dd.1" and "pm2.21dd.2" are two essential hypotheses. Similarly, an **assertion** to be proved is denoted as "[assertion name] $p [assertion] $." In Figure 3, the conclusion of the theorem pm2.21dd is "(ph -> ch)".

3. A **theorem** in general consists of several hypotheses (sometimes none) and one conclusion. In the case in Figure 3, the theorem "pm2.21dd" has two hypotheses "pm2.21dd.1" and "pm2.21dd.2" and one assertion to be proved "pm2.21dd". The theorem can be stated in natural language as follows: if $\phi$ deduces both $\psi$ and $\neg\psi$, then it deduces $\chi$. In other words, a contradiction implies anything.

4. Figure 3 show the **proving process** of the theorem "pm2.21dd" in the Metamath system. In Figure 3, the symbol "$=" in the assertion statement is followed by the proof of this theorem.

5. In the following line, "pm2.65i" and "pm2.21i" in parentheses are the axioms or proven theorems that will be referred to the proof. They are later denoted by F and G in the proof sequence, respectively. In this case, variables F and G are regarded as **operators** in the calculator, while variables A to E are **operands** in the calculator. The sequence "ACABDEFG" is the inverse Polish expression of a proof.

6. The assignment of operands and operators has the following rules: The symbol showing up first in the theorem statement is assigned first. In this case, A, B, and C denote $\phi$, $\psi$, and $\chi$. After that are the hypotheses: D denotes "pm2.21dd.1", and E denotes "pm2.21dd.2". Finally are the references: F for "pm2.65i" and G for "pm2.21i".

7. The prover **program** handles the proof process with the inverse Polish expression of the proof sequence. This means that it maintains **a proof stack** (initially empty) and uses operands and operators to push and pop the stack in turn. As shown in 3, since the first 6 elements are operands, the program simply pushes them into the stack. The seventh element (red F shown in 3) is "pm2.65i", which has four operators by its statement (2 symbols $\phi$ and $\psi$, and 2 hypotheses "pm2.65i.1" and "pm2.65i.2"). Therefore, the program pops four elements from the stack, replaces the corresponding symbols and hypotheses in the original statement ("**Substitute**:" in 3), gets the conclusion $\vdash \neg\phi$ ("**Apply**:" in 3), and pushes it into the stack (proof step 3 in 3). This process shows the only basic rule "**substitution**" in Metamath. 8. Next, the program pushes the eighth element "pm2.21i" denoted by the red G, which has three operators. The program processes it in a similar process and pushes the resulting $\phi \rightarrow \psi$ into the stack (proof step 4 in 3). At this point, the proof sequence is processed, and the only remaining elements in the stack agree with the conclusion of the theorem, which means that the theorem is successfully proved.

In Figure 6, the right part shows a brief proof of the "pm2.21dd" theorem. The first and second lines of proof refer to two hypotheses of this theorem, "pm2.21dd.1" and "pm2.21dd.2", respectively. In the third line of the proof, the theorem "pm2.65i" is referenced and the expression is shown in the left graph. "pm2.21dd.1" and "pm2.21dd.2" are used respectively to replace the original hypotheses "pm2.65i.1" and "pm2.65i.2" of the theorem "pm2.65i", and derive the conclusion $\vdash \neg\phi$. That is, if $\phi$ deduces both $\psi$ and $\neg\psi$, then $\phi$ is a false statement. The proofs in the following lines 4-7 show the process of deriving the conclusion $\vdash \phi \rightarrow \chi$ from the original set of axioms, which can be summarized as the theorem "pm2.21i", that false propositions derive everything. This case can also be seen HERE [2] for more information.

[2] https://us.metamath.org/mpeuni/pm2.21dd.html

### A.2 Details of Dataset Construction

We construct the "wb" set by selecting the first 272 theorems and then split the theorem library $\mathbb{L}'$ and problem set $\mathbb{P}$ at depth $k = 10$ and $k = 20$ for training and testing. Similarly, we construct the "wif" and "minimp" sets with $1,284$ and $2,048$ theorems, respectively. The problem set has depths larger than $k = 33$, $k = 38$ for training, and $k = 39$, $k = 40$ for testing.

### A.3 Dataset Information

The details of the statistical information of all three datasets are listed in Table 7. The resulting visualization is shown in Figure 9.

### A.4 Baseline Methods

In addition to our proposed **MCTS+pvn** trained with self-play, we also evaluate three baseline methods. The random search policy, traditional Monte Carlo tree search (MCTS), and a Byte Pair Encoding (BPE) based statistical method to find theorems given the proof in training theorem library $\mathbb{L}'$.

**Random Search** We explore the search space by expanding the most recent state with randomly selected actions, without considering their potential outcomes or evaluating their quality.

**Monte Carlo Tree Search** MCTS applies Upper Confidence Bounds (UCB) algorithm to select expanded states and actions without explicit policy probability. To obtain the value for each state-action pair, MCTS perform simulations that randomly walk to a terminal state, collect rewards and propagate back to the state-action pair and update all nodes' value in this path. We perform random walks in simulation to select the expanded nodes rather than employing a policy-value network. Specifically, we first expand the unexplored sub-nodes. If all the sub-nodes have been explored, we select the next node to expand according to the PUCT score in Equation (4) in Section 4, where $\pi(a|s)$ of all available actions are set to the same, and $v(s, a)$ is determined by the visits times of the node. During each node expansion, we obtain the node value by randomly expanding the deepest node until it reaches a terminal state. The terminal state will return a reward which is backporpated to all nodes in this path and update their value.

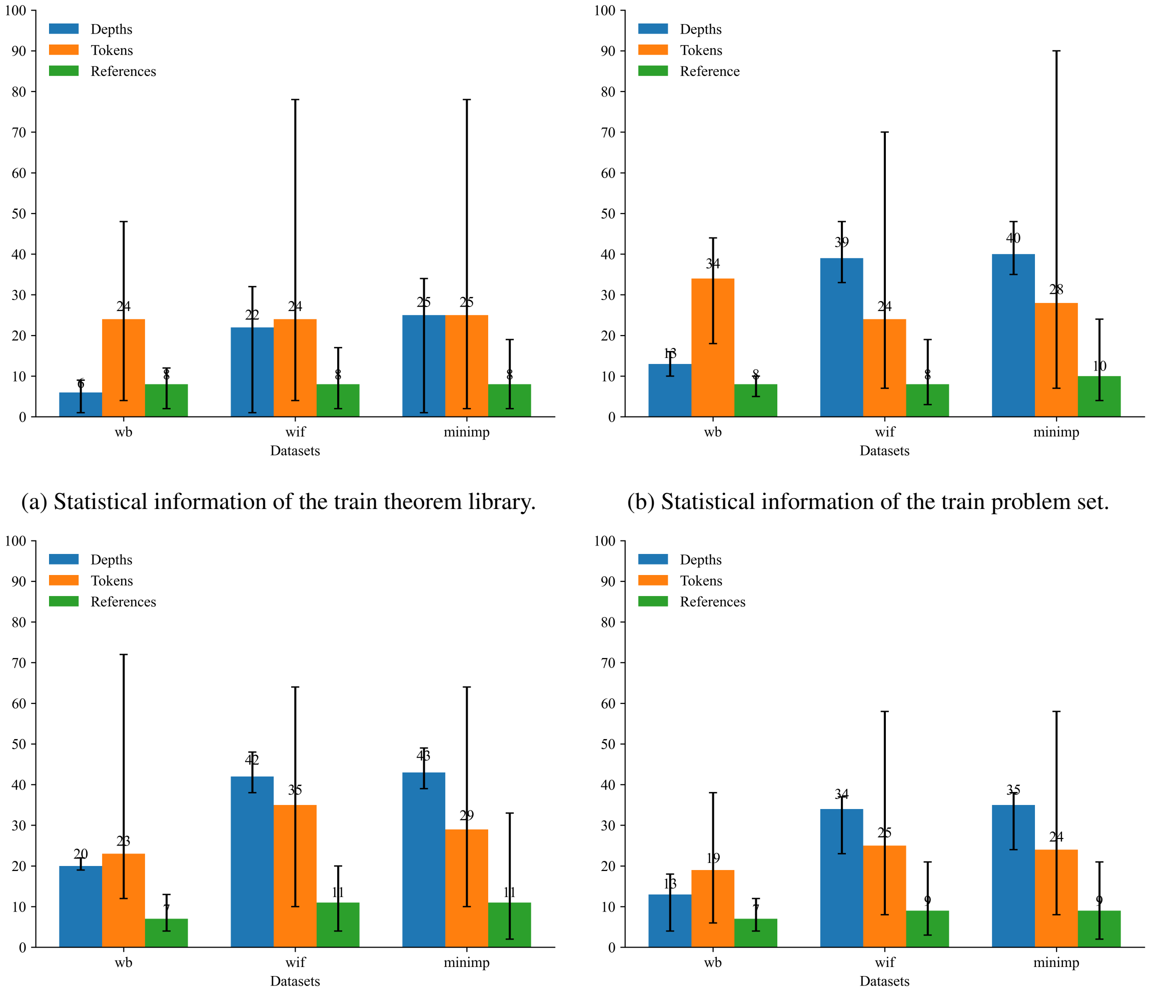
**Byte Pair Encoding** Byte Pair Encoding (BPE) is an unsupervised subword tokenization algorithm used in natural language processing. It starts with a vocabulary of characters and subwords and iteratively merges the most frequent character or subword pairs in the corpus, updating the vocabulary accordingly. In ATG, we construct vocabulary from the human-written theorems in training library $\mathbb{L}'$, each token in the proof text refers to the name of an axiom, hypothesis, or theorem, and a proof ends with a special token $[EOS]$. To obtain a valid subword, if the BPE-merged pair has an axiom or a theorem as the second item, we include previous tokens in the proof text and expand the merged subword until it is a valid proof. We locate all such pair in all training proof text and consider all resulting subwords as a generated theorem $T$. The algorithm terminates when the frequency of all token pairs is 1. Specifically, we use the full name of axioms, hypotheses, and theorems as the minimum elements (token) for grouping. Then we select the most frequent pairs in all training proofs and merge them into a new element (subword). Then we replace this pair with the new elements in all training proofs and perform merging iteratively until all pair occurrence is 1. To obtain a valid subword, if the merged pair has an axiom or a theorem as the second item, we include previous tokens in the proof and expand the merged subword until it is a valid proof. Note that the BPE method is a statistical method rather than a search algorithm, and that the source text is derived from human-written proofs, we employ it as an approximation of human-level performance.

### A.5 Implementation Details of ATG

We evaluate all of our baseline approaches with a maximum proof step length of 32 and generate 100 times in each episode, the generation ends until no more valuable theorem is generated. All of the generations begin from an initial graph that only consists of axioms and hypotheses in corresponding datasets. Methods with MCTS simulate 100 times for expanding the search tree and take a $c$ value 0.3 to balance exploration and exploitation. We collect $(state, action, value)$ triplets from proofs in the human written library $L'$ and combine data generated by MCTS policy to train the policy/value network in each episode. The training process contains 10 epochs per episode and takes 128 samples per batch. The learning rate is set to $3e - 4$ and we train our models with the Adam optimizer. We also apply an early-stop mechanism at minimum KL divergence between policy probabilities before and after training.

| Dataset | Depth | | | Token | | | References | | |
|---|---|---|---|---|---|---|---|---|---|
| | min | avg | max | min | avg | max | min | avg | max |
| wb train library | 1 | 6 | 9 | 4 | 24 | 48 | 2 | 8 | 12 |
| wb train problem | 10 | 13 | 16 | 18 | 34 | 44 | 5 | 8 | 10 |
| wb test library | 4 | 13 | 18 | 6 | 19 | 38 | 4 | 7 | 12 |
| wb test problem | 19 | 20 | 22 | 12 | 23 | 72 | 4 | 7 | 13 |
| wif train library | 1 | 22 | 32 | 4 | 24 | 78 | 2 | 8 | 17 |
| wif train problem | 33 | 39 | 48 | 7 | 24 | 70 | 3 | 8 | 19 |
| wif test library | 23 | 34 | 37 | 8 | 25 | 58 | 3 | 9 | 21 |
| wif test problem | 38 | 42 | 48 | 10 | 35 | 64 | 4 | 11 | 20 |
| minimp train library | 1 | 25 | 34 | 2 | 25 | 78 | 2 | 8 | 19 |
| minimp train problem | 35 | 40 | 48 | 7 | 28 | 90 | 4 | 10 | 24 |
| minimp test library | 24 | 35 | 38 | 8 | 24 | 58 | 2 | 9 | 21 |
| minimp test problem | 39 | 43 | 49 | 10 | 29 | 64 | 2 | 11 | 33 |

Table 7: Detailed statistics of wb, wif, minimp datasets in the ATG benchmark.



(a) Statistical information of the train theorem library.

(b) Statistical information of the train problem set.

(c) Statistical information of the test theorem library.

(d) Statistical information of the test problem set.

Figure 9: The average, min, max numbers of proof Depth, tokens and referred theorems of all three datasets.

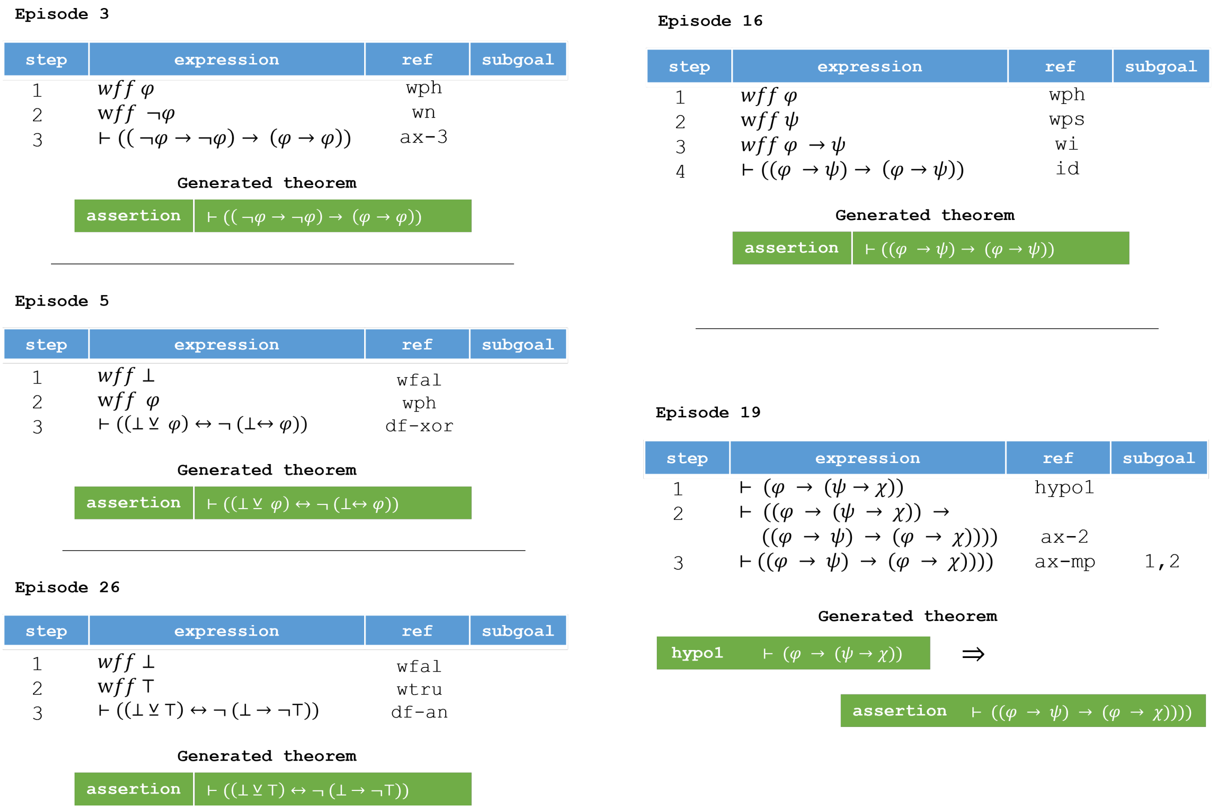
### A.6 Implementation Details of ATP

We test two provers: the Holophrasm prover and the GPT-f prover. Because different provers use different versions of "set.mm" libraries, the data pre-processing and the training pipeline vary accordingly. The generated theorems are then formatted into standard Metamath language form as augmented data. We further use the BPE method to generate more theorems based on the "minimp" dataset, and then randomly substitute the variables in the theorems to construct an augmented theorem sets with $\sim 30k$ proof steps.

During evaluation, we follow the original Holophrasm prover (Whalen, 2016) and the original GPT-f prover (Polu and Sutskever, 2020) setups and use the same data splits.

### A.7 More Examples of Generated Theorems

Figure 10 provides more theorems generated by MCTS with policy/value network on the "minimp" dataset, these theorems are referred when proving other theorems in downstream ATP tasks. Figure 11 shows more complex theorems generated by our method. These theorems involve complex concepts in propositional logic such as the distribution of implication over biconditional and require as many as dozens of proof steps.



**Episode 3**

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $wff\ \varphi$ | wph | |
| 2 | $wff\ \neg\varphi$ | wn | |
| 3 | $\vdash ((\neg\varphi \to \neg\varphi) \to (\varphi \to \varphi))$ | ax-3 | |

Generated theorem

| assertion | $\vdash ((\neg\varphi \to \neg\varphi) \to (\varphi \to \varphi))$ |
|---|---|

**Episode 5**

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $wff\ \bot$ | wfal | |
| 2 | $wff\ \varphi$ | wph | |
| 3 | $\vdash ((\bot \veebar \varphi) \leftrightarrow \neg(\bot \leftrightarrow \varphi))$ | df-xor | |

Generated theorem

| assertion | $\vdash ((\bot \veebar \varphi) \leftrightarrow \neg(\bot \leftrightarrow \varphi))$ |
|---|---|

**Episode 26**

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $wff\ \bot$ | wfal | |
| 2 | $wff\ \top$ | wtru | |
| 3 | $\vdash ((\bot \veebar \top) \leftrightarrow \neg(\bot \to \neg\top))$ | df-an | |

Generated theorem

| assertion | $\vdash ((\bot \veebar \top) \leftrightarrow \neg(\bot \to \neg\top))$ |
|---|---|

**Episode 16**

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $wff\ \varphi$ | wph | |
| 2 | $wff\ \psi$ | wps | |
| 3 | $wff\ \varphi \to \psi$ | wi | |
| 4 | $\vdash ((\varphi \to \psi) \to (\varphi \to \psi))$ | id | |

Generated theorem

| assertion | $\vdash ((\varphi \to \psi) \to (\varphi \to \psi))$ |
|---|---|

**Episode 19**

| step | expression | ref | subgoal |
|---|---|---|---|
| 1 | $\vdash (\varphi \to (\psi \to \chi))$ | hypo1 | |
| 2 | $\vdash ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))))$ | ax-2 | |
| 3 | $\vdash ((\varphi \to \psi) \to (\varphi \to \chi))))$ | ax-mp | 1,2 |

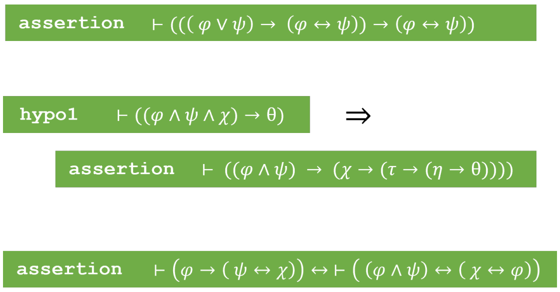
Generated theorem

| hypo1 | $\vdash (\varphi \to (\psi \to \chi))$ |
|---|---|

$\Rightarrow$

| assertion | $\vdash ((\varphi \to \psi) \to (\varphi \to \chi))))$ |
|---|---|

Figure 10: Some generated theorems are referred to in downstream ATP tasks.



| assertion | $\vdash (((\varphi \vee \psi) \to (\varphi \leftrightarrow \psi)) \to (\varphi \leftrightarrow \psi))$ |
|---|---|

| hypo1 | $\vdash ((\varphi \wedge \psi \wedge \chi) \to \theta)$ |
|---|---|

$\Rightarrow$

| assertion | $\vdash ((\varphi \wedge \psi) \to (\chi \to (\tau \to (\eta \to \theta))))$ |
|---|---|

| assertion | $\vdash (\varphi \to (\psi \leftrightarrow \chi)) \leftrightarrow \vdash ((\varphi \wedge \psi) \leftrightarrow (\chi \leftrightarrow \varphi))$ |
|---|---|

Figure 11: Example complex theorems generated by the proposed MCTS+pvn.