Usefulness-Driven Learning of Formal Mathematics

Timothe Kasriel
UC Berkeley
tkasriel@berkeley.edu

Thomas Lu
UC Berkeley
thomaslu@berkeley.edu

Devon Ding
UC Berkeley
devon_ding@berkeley.edu

Jingxuan He
UC Berkeley
jingxuan.he@berkeley.edu

Dawn Song
UC Berkeley
dawnsong@berkeley.edu

Abstract

Creating an AI that can truly "do" mathematics requires more than just solving isolated problems: it must be able to progressively build up a corpora of useful knowledge in a similar manner as do mathematicians. To this end, we introduce UseFor, a novel framework to formalize this notion of building up knowledge, and demonstrate how it can be used to train a usefulness-driven AI mathematician. UseFor determines a theorem's usefulness based on two criteria: its reusability in subsequent proofs and its contribution to increasing proof likelihood. We integrate UseFor into the self-play setting of Minimo ([10]), training a model from scratch through a conjecturing and proving self-play loop with usefulness testing. We experimentally evaluate this usefulness-driven self-play approach across three mathematical domains: arithmetic, propositional logic, and group theory with two metrics: intrinsic usefulness, a measure of how often the lemmas are used, and extrinsic usefulness, a measure driven by LLM evaluation. Our results demonstrate that our usefulness-trained model effectively generates a large number of intrinsically and extrinsically useful formal theorems.

1 Introduction

While large language models (LLMs) have achieved rapid progress in formal theorem proving [18], most approaches depend on extensive human-written corpora of proofs and conjectures [19, 20]. Further, mathematical research involves far more than proving problems alone, requiring accumulated knowledge of a domain and the ability to determine which problems are worth pursuing. As such, we propose the following: Can an artificial agent, starting only from axioms, learn via self-play between conjecturing and proving, bootstrapping its own knowledge and progressively discovering new mathematics? This paradigm [8] would reduce the need for human data, enable exploration of domains where no proofs exist, and produce a scalable source of synthetic data for training models.

Inspired by the perspective [14, 2] that a theorem should be judged not in isolation but with respect to its applicability to other problems, we propose UseFor, a tractable metric for evaluating the usefulness of mathematical conjectures. UseFor contributes most directly to the problems of *premise discovery* [17] and *theory exploration* [7]. (We discuss related work in Appendix A). We leverage UseFor to develop a usefulness-aware self-play loop that builds directly on Minimo's [10] self-play loop of conjecturing and proving from minimal axioms. By training the model on an extra step of usefulness testing, we guide the model toward structures that accelerate cumulative theory building.

We evaluate our usefulness-driven self-play framework on three domains: arithmetic, propositional logic, and group theory. Our results show that UseFor allows our model to learn to leverage previously

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

proven theorems in proofs, and generate significantly more useful conjectures compared to baseline Minimo, as evaluated by LLM-as-a-judge.

Main contributions. Our key contributions are: (i) We formalize the notion of theorem usefulness as a dual criterion of *usage* and *improvement*, and propose a tractable procedure for measuring it within self-play. (ii) We introduce a usefulness-aware self-play loop that augments Minimo by selecting conjectures according to relational usefulness rather than difficulty. (iii) We provide empirical results across arithmetic, propositional logic, and group theory, demonstrating that usefulness-driven conjectures are more reusable and lead to higher prover success rates than difficulty-based baselines.

2 Methodology

2.1 Base Self-Play Framework (Minimo)

Conjecturing. At the base of Minimo is a self-play loop which begins with conjecturing. The conjecturer $\mathcal{C}_\theta$, with θ denoting the model parameters, generates statements in the Peano language, a dependently typed formal language [11] with finite action space. To prevent invalid formulas, Minimo employs constrained decoding [12]: at each step, candidate tokens are filtered so that only those extending the current tree into a valid continuation remain.

Proving. Once the conjectures $\{c_0, c_1, ..., c_n\}$ are established, the prover $\mathcal{P}_\theta$ attempts prove these conjectures using a Monte Carlo tree search (MCTS). Guided by the prover's policy $\pi_\theta(a \mid s)$ on the proof state s action a, and value estimates, MCTS expands trajectories $\tau = (s_0, a_0, s_1, s_2, ..., s_n, a_n)$ starting from the initial state s_0. Which can be scored by their log-likelihood $\ell(c)$ under the prover's policy, with a less negative indicating a higher prover certainty. To exploit partial progress, Minimo applies *hindsight relabeling*: even when a conjecture cannot be proved in full, explored search trees are decomposed into valid subtraces corresponding to intermediate lemmas, which are then incorporated as additional training data [10].

Self-Play Loop. The conjecturer $\mathcal{C}_{\theta_i}$ and prover $\mathcal{P}_{\theta_i}$ interact in an iterative loop. At iteration i, the conjecturer samples a batch of N candidate statements $\mathcal{Q}_i = \{c_1, \ldots, c_N\} \sim \mathcal{C}_{\theta_i}(\cdot \mid \mathcal{T}_i)$, where $\mathcal{T}_i$ is the current theory consisting of axioms and previously promoted lemmas. For each $c \in \mathcal{Q}_i$, the prover attempts to establish it via MCTS: $(\text{proof}(c),\ \ell(c),\ \text{trace}(c)) \leftarrow \texttt{MCTS\_PROVE}(c;\ \mathcal{T}_i,\ \mathcal{P}_{\theta_i})$, where $\text{proof}(c)$ is a complete proof trajectory τ if one is found (or $\varnothing$ otherwise), $\text{trace}(c)$ is the explored search tree, and $\ell(c)$ is the log-likelihood of the trajectory under the prover's policy.

The conjectures $\mathcal{Q}_i$ are then stratified by empirical difficulty, as measured by proof log-probability: conjectures below the 50th percentile of difficulty are labeled as "trivial", the top 20th percentile as "hard", and others as "easy". The combined prover-conjecturer model is then trained on the results of our iteration $\mathcal{E}_i$:

$$\mathcal{E}_i = \{(\text{trace}(c),\ \text{label}(c)) : c \in \mathcal{Q}_i\},$$

which aggregates conjectures, proofs when available, and hindsight-relabeled subproofs extracted from failed searches. As conjectures for which proving has failed cannot be labeled, we instead extract proof examples from trace(c) and add to $\mathcal{E}_i$.

2.2 Usefulness-Aware Self-Play Loop

The self-play framework of Minimo provides a compelling basis for data-free theory exploration. However, its training signal is limited to conjectural difficulty, measured as the negative log-probability of a proof under the current prover. As a result, the system often promotes conjectures that are labeled as "hard" but not necessarily useful statements: isolated identities that stretch the prover temporarily but are rarely reused and add little structure to the theory.

To address this limitation, we introduce a usefulness-based self-play loop. We ask whether incorporating a new lemma makes other statements easier to prove. Conjectures that are both provable and demonstrably beneficial in downstream proofs are promoted into the growing library.

UseFor: Our Usefulness Metric. The perspectives of Benigo et. al [2] and Tao [14] converge on the idea that the value of a theorem is relational: it derives its significance not from truth alone, but

from its effect on subsequent reasoning. Benigo et. al [2] frames usefulness in information-theoretic terms, proposing that a theorem acts as a *compression primitive*—its addition to a base theory reduces the description length of other proofs. Tao [14] instead emphasizes the pragmatic dimension: the *strength* of a theorem is revealed only by confronting new problems and observing the range of arguments it simplifies.

Our contribution is to construct a practical formulation of these metrics. Formally, let $\mathcal{B}$ be a benchmark set consisting of theorems that are difficult, but not impossible, for the prover to prove. For each $b \in \mathcal{B}$, let $p_\theta(\tau_b \mid b)$ denote the prover's probability of producing a proof trajectory τ_b under theory $\mathcal{T}$, and let $p'_\theta(\tau_b \mid b)$ denote the same quantity when a candidate lemma ℓ is available. We say that ℓ is *useful* if there exists $b \in \mathcal{B}$ such that

$$\text{(i) } \ell \text{ is invoked in the proof trace of } b, \qquad \text{and} \qquad \text{(ii) } \log p'_\theta(\tau_b \mid b) - \log p_\theta(\tau_b \mid b) > 0.$$

Both conditions are essential: usage without improvement admits trivial tautologies such as $\forall x.\, x = x$, which the prover may frequently attempt but which yield no real progress. To evaluate this metric, given a set of previously proven lemmas $\mathcal{C}$, we subsample a subset of size $\lceil \sqrt{|\mathcal{C}|} \rceil$ and temporarily add them to the context. Each $b \in \mathcal{B}$ is then reproven once under this extended theory. If a candidate $\ell \in \mathcal{C}$ appears in the proof of b and the resulting log-likelihood improves relative to baseline, the gain is attributed to ℓ. Their total log probability increase (ii) is then used to rank the lemmas, and promote them in the library. This provides a tractable mechanism for selecting conjectures that repeatedly demonstrate both reuse and measurable downstream gains.

Training Loop with UseFor. We now describe how the usefulness metric is integrated into the conjecturing–proving loop. The outer structure mirrors Minimo [10]: in each iteration the agent generates conjectures, the prover attempts proofs via MCTS, and traces are collected. The crucial difference lies in how conjectures are filtered, promoted, and fed back into training. At iteration i, the conjecturer first proposes a batch $\mathcal{C}_i$. During conjecturing, we penalize the conjecturer for repeating previously generated prefixes, encouraging diversity and avoiding local minima. Trivial tautologies (ex: $x = x$) are also removed using heuristic filters. Each conjecture is then attempted under the current theory $\mathcal{T}_i$ using MCTS, producing proofs, log-likelihoods, and hindsight examples. Following Minimo, conjectures are provisionally bucketed into "hard," "easy," and "trivial" categories by percentile of log-likelihood.

The key departure comes in how the "hard" subset is treated. Rather than promoting them directly, we apply the *usefulness test* using the current pool of non-failing lemmas as context. A random subsample $L_i \subseteq \mathcal{H}_i$ of size $\lceil \sqrt{|\mathcal{H}_i|} \rceil$ is drawn, and each benchmark $b \in \mathcal{B}$ is reproven under both the baseline theory $\mathcal{T}_i$ and the augmented theory $\mathcal{T}_i \cup L_i$. If a lemma $\lambda \in L_i$ is invoked in the augmented proof of b and improves its log-likelihood relative to the baseline, the gain is added to its cumulative usefulness score $U_i(\lambda)$. Candidates are then ranked by $U_i(\lambda)$, and only the top ρ fraction are promoted into the persistent theory $\mathcal{T}_{i+1}$. Because L_i is resampled at every iteration, different subsets of candidates are tested over time, so all conjectures eventually receive usefulness credit.

Finally, the training dataset $\mathcal{E}_i$ is assembled. It contains the conjectures, their proofs, and percentile labels, and the hindsight traces from the base loop, as well as the useful lemmas and re-proving examples produced during usefulness testing. The agent is updated on $\mathcal{E}_i$, and the promoted lemmas are added to $\mathcal{T}_{i+1}$ for future iterations.

3 Experimental Evaluation

We evaluate UseFor on three domains: arithmetic, propositional logic, and group theory. The axioms for each domain are listed in Appendix C. Due to space constraints, we only showcase the results for arithmetic in this section. Results for propositional-logic and group theory can be found in Appendix E. Our goal is to assess whether UseFor demonstrates the essential qualities of a desirable reasoning system: (a) the ability to accumulate knowledge across iterations, (b) the ability to generate conjectures of intrinsic value, and (c) whether our usefulness-driven training is necessary. We employ two complementary metrics designed to capture structural usefulness:

- *Intrinsic usefulness*: measured as the number of times a previously proven theorem is reused during usefulness testing. A high score indicates that the system is both conjecturing and successfully reusing theorems in its own proving process.

- *Extrinsic usefulness*: measured via an LLM-as-judge (details can be found in Appendix D), which rates conjectures for mathematical value. These conjectures are additionally proven by an external prover based on the Z3 SMT solver [4]. This metric evaluates whether conjectures would be judged useful by a human mathematician, beyond the system's internal dynamics.

Details about experimental setup are in Appendix B

(RQ1) Can the prover reuse theorems proven in previous iterations to prove current conjectures? Reuse is essential for cumulative theory building: without it, a system risks repeatedly rediscovering tautologies or isolated results. In our experiments, UseFor shows a steady increase in lemma usage during usefulness testing (Figure 1). Although the first few iterations provide little signal, usage accelerates in later iterations, demonstrating that the model progressively conjectures more useful theorems and becomes increasingly capable of applying them. This trend is consistent across all domains, and we expect it to persist with additional iterations. As MINIMO's method does not include usefulness testing and so does not have access to previously proven theorems, its intrinsic metric is 0, will only be included as a baseline in RQ2 and RQ3.

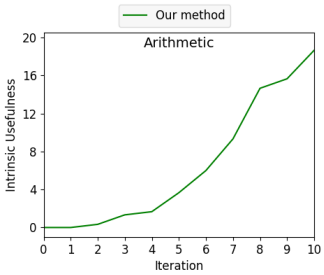

Figure 1: Intrinsic Evaluation: Total theorem usages.

(RQ2) Do likelihoods of theorem-reusing proofs increase across multiple iterations? Here we evaluate whether the prover gains more capabilities and confidence in theorem reuse during the training progress. Figure 2 shows that the average likelihood of proofs where previous conjectures were used consistently increases across the training process. This aligns with the significant increase in intrinsically useful conjectures across multiple iterations, as shown in Figure 1, and shows that the UseFor training objective is effective in encouraging theorem reuse.

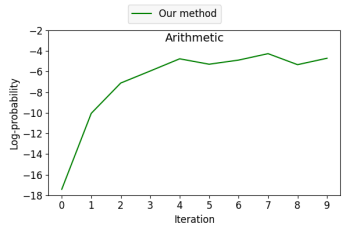

Figure 2: Mean log-probabilities of proofs in which a previously conjecture was used.

(RQ3) Are the conjectures useful beyond self-play? Extrinsic usefulness (Figure 3) tests whether the system discovers theorems a human mathematician would value (simulated by LLM-as-a-judge and an SMT solver in our evaluation), beyond the self-play training loop itself. In early iterations, UseFor quickly identifies many "easy" theorems accessible through shallow search. Usefulness continues to increase in later iterations, indicating that the system discovers progressively deeper and less trivial results. In Appendix D.4, we provide examples of extrinsically useful theorems conjectured by our model.

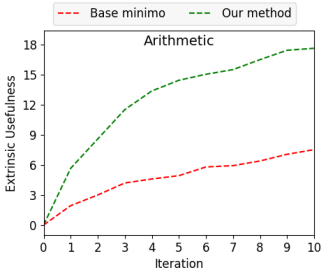

Figure 3: Extrinsic Evaluation: Number of useful theorems judged by GPT-4.1 and Z3 solver.

(RQ4) Is the usefulness metric essential for both conjecturing and proving? This experiment evaluates how our usefulness training signal affects performance (Figure 4). As seen in Figure 4, if training is omitted, the system performs markedly worse: extrinsically, fewer theorems are judged to be useful by LLM-as-a-judge and SMT solver. This demonstrates the importance of training for updating the conjecturer with usefulness feedback steers it toward generating conjectures that are genuinely valuable for future proofs.

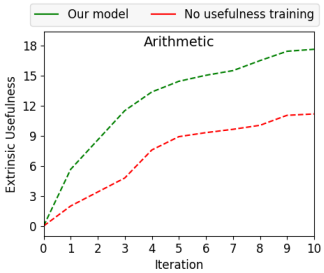

Figure 4: Ablation showing the necessity of usefulness training.

4 Conclusion

Across considered domains, UseFor increases both the intrinsic reuse of conjectured theorems and the extrinsic usefulness of its discoveries, confirming that usefulness-aware self-play can build coherent and cumulative theories directly from axioms. However, our study is confined to relatively

small models, limited axioms, and fixed search budgets; scaling to richer foundations (e.g., Lean, Isabelle) and larger models remains an open challenge. Applying approaches like UseFor on bigger models, such as large language models pretrained large corpora, brings the additional risk of data contamination, and so future work must find ways to distinguish between truly novel conjectures and ones duplicated from the pretraining corpus in the setting where LLMs are used for conjecturing new mathematical theorems.

References

[1] Yousef Alhessi, Sólrún Halla Einarsdóttir, George Granberry, Emily First, Moa Johansson, Sorin Lerner, and Nicholas Smallbone. Lemmanaid: Neuro-symbolic lemma conjecturing, 2025.

[2] Yoshua Bengio and Nikolay Malkin. Machine learning and information theory concepts towards an ai mathematician. *Bulletin of the American Mathematical Society*, 61(3):457–469, 2024.

[3] Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Yuchen Wu, Yihang Xia, Huajian Xin, Fan Yang, Huaiyuan Ying, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang, Ge Zhang, Tianyun Zhao, Jianqiu Zhao, Yichi Zhou, and Thomas Hanwen Zhu. Seed-prover: Deep and broad reasoning for automated theorem proving, 2025.

[4] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[5] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025.

[6] Thibault Gauthier and Josef Urban. Learning conjecturing from scratch, 2025.

[7] Moa Johansson and Nicholas Smallbone. Conjectures, tests and proofs: An overview of theory exploration. *Electronic Proceedings in Theoretical Computer Science*, 341:1–16, September 2021.

[8] David McAllester. Mathzero, the classification problem, and set-theoretic type theory, 2020.

[9] Naoto Onda, Kazumi Kasaura, Yuta Oriike, Masaya Taniguchi, Akiyoshi Sannai, and Sho Sonoda. Leanconjecturer: Automatic generation of mathematical conjectures for theorem proving, 2025.

[10] Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation, 2024.

[11] Gabriel Poesia and Noah D. Goodman. Peano: learning formal mathematical reasoning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251), June 2023.

[12] Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations (ICLR)*, 2021.

[13] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020.

[14] Terence Tao. On the strength of theorems. `https://terrytao.wordpress.com/advice-on-writing-papers/on-the-strength-of-theorems/`, 2007. Accessed: 2025-09-15.

[15] Josef Urban and Jan Jakubův. First neural conjecturing datasets and experiments, 2020.

[16] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Heng Liao, and Xiaodan Liang. Lego-prover: Neural theorem proving with growing libraries, 2023.

[17] Yutong Xin, Jimmy Xin, Gabriel Poesia, Noah Goodman, Qiaochu Chen, and Isil Dillig. Automated discovery of tactic libraries for interactive theorem proving, 2025.

[18] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai, 2024.

[19] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models, 2023.

[20] Huaiyuan Ying, Zijian Wu, Yihan Geng, Zheng Yuan, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems, 2025.

A Related Work

Our work is primarily related to prior bodies of work on mathematical conjecturing, tactic discovery, and theory exploration. Our approach is distinguished by the fact that our model is trained in a tabula rasa fashion, without any pre-existing examples, and evaluated on the theory exploration task.

Mathematical conjecturing. Our work is most closely based on Minimo [10], which proposes a theorem-proving model in the Peano [11] formal language that is trained through iterative conjecturing and proving from scratch. [13] also propose a model that is trained via self-play, while [5] demonstrate the ability of the iterative conjecturing-proving paradigm to enhance a pretrained theorem prover. However, these works only use conjecturing as a means to improve the proof-search capabilities of the model, and do not attempt to evaluate the conjecturing abilities of the model directly. LeanConjecturer [9] proposes a model specifically designed for the conjecturing task, but uses a pretrained LLM; in doing so, the ability of the LeanConjecturer model to generate novel conjectures cannot be faithfully evaluated due to inevitable contamination from pre-training data. Compared to these works, our approach evaluates conjecturing as a stand-alone task, while our tabula rasa setting allows us to definitively confirm the novelty of conjectures generated by our model.

Tactic and premise discovery. There is also a body of work concerning the task of tactic discovery, which aims to construct tactics in an interactive theorem prover setting that simplify proofs or otherwise enhance proving capabilities. TacMiner [17] proposes a method to find tactic simplifications in RCoq, given an existing high-quality corpus of proofs. Lego-Prover [16] and Seed-Prover [3] use already proven lemmas as a way to strengthen a theorem proving model, in the Isabelle and Lean 4 settings, respectively. However, all of these approaches require a dataset of high-quality, human-generated proofs, while our approach generates useful premises from scratch.

Theory exploration using machine learning. Finally, a third body of work is theory exploration using ML methods, the task of formulating interesting conjectures about a given problem domain [7]. We consider this problem to be the one our work addresses most closely. While a number of classical and neural approaches have been proposed for this task, existing neural methods work by training or finetuning a model based on an existing proof corpus [15]. Lemmanaid [1] uses neuro-symbolic methods by finetuning a model with a subset of an existing proof library, and then evaluating it on another subset of conjectures. In search of a purely intrinsic approach in order to discover how a model could discover this usefulness without relying on human data, we distinguish ourselves by not training on external data, an approach similar to what has been done for SMT solvers [6].

B Details about Experimental Setup

All models were trained for 10 iterations, with 200 conjectures generated per iteration. Each experiment was repeated three times, and we report mean values with standard deviations across runs. Proof search was conducted using Monte Carlo Tree Search (MCTS) with a budget of 1000

expansions per conjecture. We repeat all experiments for three times and report averaged results to account for stochastic variations.

C Axioms

We now provide all the axioms for the three domains considered in our experiments in Section 3. They are taken from the Minimo paper [10] and formalized in the Peano languages [11].

Arithmetic

```
= : [nat -> nat -> prop].
nat : type.

z : nat.
s : [nat -> nat].
o : nat.

+ : [nat -> nat -> nat].
* : [nat -> nat -> nat].

o_s : (= o (s z)).

+_z : [('n : nat) -> (= (+ 'n z) 'n)].
+_s : [('n : nat) -> ('m : nat) -> (= (+ 'n (s 'm)) (s (+ 'n 'm)))].

*_z : [('n : nat) -> (= (* 'n z) z)].
*_s : [('n : nat) -> ('m : nat) -> (= (* 'n (s 'm)) (+ 'n (* 'n 'm)))].

nat_ind : [('p : [nat -> prop]) -> ('p z) -> [('n : nat) ->
          ('p 'n) -> ('p (s 'n))] -> [('n : nat) -> ('p 'n)]].

#backward nat_ind.
#forward +_z ((+ 'n z) : nat).
#forward +_s ((+ 'n (s 'm)) : nat).
#forward *_z ((* 'n z) : nat).
#forward *_s ((* 'n (s 'm)) : nat).
```

Propositional logic

```
prop : type.

false : prop.

/* Connectives */
not : [prop -> prop].
and : [prop -> prop -> prop].
or : [prop -> prop -> prop].
iff : [prop -> prop -> prop].

/* Introduction rule for conjunction */
#backward and_i.
and_i : [('P : prop) -> ('Q : prop) -> 'P -> 'Q -> (and 'P 'Q)].
/* Elimination rules for conjunction */
#forward and_el ('_ : (and 'P 'Q)).
and_el : [('P : prop) -> ('Q : prop) -> (and 'P 'Q) -> 'P].
#forward and_er ('_ : (and 'P 'Q)).
and_er : [('P : prop) -> ('Q : prop) -> (and 'P 'Q) -> 'Q].

/* Introduction rules for disjunction */
#backward or_il.
or_il : [('P : prop) -> ('Q : prop) -> 'P -> (or 'P 'Q)].
#backward or_ir.
or_ir : [('P : prop) -> ('Q : prop) -> 'Q -> (or 'P 'Q)].
```

```
/* Elimination rule for disjunction */
#backward or_e infer infer infer infer subgoal subgoal.
or_e : [('P : prop) -> ('Q : prop) -> ('R : prop) ->
       (or 'P 'Q) -> ['P -> 'R] -> ['Q -> 'R] -> 'R].

/* Introduction rule for negation */
#backward not_i.
not_i : [('P : prop) -> ['P -> false] -> (not 'P)].
/* Elimination rule for negation */
not_e : [('P : prop) -> (not 'P) -> 'P -> false].
#backward exfalso.
exfalso : [false -> ('P : prop) -> 'P].

/* Introduction rules for equivalence */
#backward iff_i.
iff_i : [('P : prop) -> ('Q : prop) -> ['P -> 'Q] -> ['Q -> 'P] -> (iff 'P 'Q)].
/* Elimination rules for equivalence */
#forward iff_el ('_ : (iff 'P 'Q)).
iff_el : [('P : prop) -> ('Q : prop) -> (iff 'P 'Q) -> ['P -> 'Q]].
#forward iff_er ('_ : (iff 'P 'Q)).
iff_er : [('P : prop) -> ('Q : prop) -> (iff 'P 'Q) -> ['Q -> 'P]].

/* Excluded middle */
#forward em.
em : [('P : prop) -> (or 'P (not 'P))].
```

Group theory

```
= : [('t : type) -> 't -> 't -> prop].

G : type.

op : [G -> G -> G].
id : G.

/* Associativity */
#forward op_assoc ((op (op 'a 'b) 'c) : G).
op_assoc : [('a : G) -> ('b : G) -> ('c : G) ->
           (= (op (op 'a 'b) 'c) (op 'a (op 'b 'c)))].

/* Commutativity */
#forward op_comm ((op 'a 'b) : G).
op_comm : [('a : G) -> ('b : G) -> (= (op 'a 'b) (op 'b 'a))].

/* Identity */
#forward id_l.
id_l : [('a : G) -> (= (op id 'a) 'a)].

/* Inverse */
inv : [G -> G].
#forward inv_l.
inv_l : [('a : G) -> (= (op (inv 'a) 'a) id)].
```

D Extrinsic Evaluation

In order to perform extrinsic evaluation, we run 5 iterations of our extrinisc evaluation pipeline, and take the average of the 5 results in order to mitigate variance from different runs of LLM evals. Our extrinsic evaluation pipeline consists of two steps: usefulness checking (Appendix D.1), deduplication (Appendix D.2), and SMT solving. In usefulness checking, we prompt the model concurrently on all conjectures generated by the model and keep the ones marked as useful by the LLM. As we are concurrently requesting for usefulness, we are likely to get a large amount of duplicate conjectures. We therefore make a second pass, calling the model on the useful conjectures to deduplicate them,

keeping only sufficiently different theorems so as to get more reasonable results. Finally, we leverage the Z3 SMT solver [4] to automatically prove the remaining conjectures and count only the proven ones. We found Z3 to be highly effective in proving these conjectures, as they are derived from axioms.

In the specific case of group theory, we noticed the variance in LLM evaluations was significantly higher than other domains, and the LLM had a very high rate of returning false problems. We solved this by running the SMT solver first, and giving a custom deduplication prompt (Appendix D.3) with examples for group theory.

D.1 Usefulness Checking Prompt

```
You are tasked to judge whether a given lean theorem could be considered useful for
    an automatic theorem prover to have among its known theorems.
This theorem prover has only access to the following axioms and known theorems:
‘‘‘
‘‘‘
{known_theorems}
As well as access to the ‘rfl‘ and ‘rewrite‘ commands
Here is the theorem you are to evaluate
‘‘‘lean4
{generated_conjecture}
‘‘‘
Think through the problem step by step. Translate the problem into natural language,
     then think of what the possible uses of the theorem could be, whether it’s
    obviously true and whether it means something.
On the last line, say either USEFUL or NOT USEFUL and nothing else.
```

D.2 Deduplication Prompt

```
I have a set of lean theorems, some of which are very similar to each other. I want
    to use them as tactics for proof generation.
Please remove the duplicates, so that I can have a list of only unique theorems.
For example, the following four theorems would be duplicates of each other:
‘‘‘lean4
theorem problem1 : (v0 : Nat) -> v0 * 1 = v0
theorem problem2 : (v0 : Nat) -> (v1 : Nat) -> v1 * 1 = v1
theorem problem3 : (v0 : Nat) -> (v1 : Nat) -> (v2 : v0 = v1) -> v1 * 1 = v1
theorem problem4 : (v0 : Nat) -> v0 * (Nat.succ 0) = v0
‘‘‘
The inclusion of an extra variable in problem 2 doesn’t change the fact that the
    result is exactly the same, and the different names for the variable doesn’t
    affect the result.
Problem 3 introduces an irrelevant hypothesis, which doesn’t get used in the theorem
    , and the conclusion is still the same.
The last one is a trivial result of the others, as 1 is defined as Nat.succ 0 in
    this case.
Here is my list of theorems for you to remove duplicates for.
{}
I also have attached an explanation for why each could be useful for a theorem
    prover.
{}
Think it through step by step, and then return the list of unique theorems from this
     list in a list format inside of a ‘‘‘lean4‘‘‘ code block. Make sure your
    answer is inside the very last lean codeblock. Please make sure to repeat the
    theorems exactly as I wrote them.
```

D.3 Group Theory Specific Prompts

```
I have a set of lean theorems, some of which are very similar to each other. I want
    to use them as lemmas for proof generation.
Please remove the duplicates, so that I can have a list of only unique theorems.
```

```
For example, the following four theorems would be duplicates of each other:
‘‘‘lean4
theorem problem1 : ((v0 : Group) -> (v1 : (v0 = (v0 * (1^{-1})))) -> ((1^{-1}) = 1))
theorem problem2 : ((v0 : Group) -> (v1 : Group) -> ((1^{-1}) = 1))
theorem problem3 : ((v0 : Group) -> ((1^{-1}) = 1))
theorem problem4 : ((v0 : Group) -> (1 = (1^{-1}))
‘‘‘
Problem 1 introduces an irrelevant hypothesis as compared to problem 3, as it makes
    no mention of v0 in its final claim. Therefore, these two problems are
    duplicates of each other.
Problem 2 is a similar case to problem 1: It introduces an extra variable, but does
    nothing with it. This is irrelevant, and makes for the same problem.
Problem 4 is the same as problem 3, but is flipped. As we are running this using rw,
     we can simply call this problem in the inverse direction, so these two lemmas
    are the same.

In this case, our final result would likely be:
‘‘‘lean4
theorem problem3 : ((v0 : Group) -> ((1^{-1}) = 1))
‘‘‘

Here is my list of theorems for you to remove duplicates for.
{}
I also have attached an explanation for why each could be useful for a theorem
    prover.
{}
Think it through step by step, and then return the list of unique theorems from this
     list in a list format inside of a ‘‘‘lean4‘‘‘ code block. Make sure your
    answer is inside the very last lean codeblock. Please make sure to repeat the
    theorems exactly as I wrote them.
```

D.4 Examples of Extrinsically Useful Conjectures

Table 1 highlights representative conjectures that our evaluation judged to be extrinsically useful across three domains. These serve as concrete examples of the kinds of results UseFor is capable of producing. As an illustration, UseFor produces a 5-step proof of the first propositional-logic conjecture in Table 1, using only the base axioms. However, given the tactic `iff_elim`, which reduces an equivalence to two implications, together with the axioms

$$False \implies P, \tag{1}$$
$$P \wedge Q \implies P, \tag{2}$$

UseFor found the following 5-step proof:

1.Split the problem into cases: by `iff_elim`
- Case 1: $False \implies P \wedge False$
  - 2.introduce $False$ into hypothesis context
  - 3.$False \implies P \wedge False$ by (1)
- Case 2: $P \wedge False \implies False$
  - 4.introduce $P \wedge False$ into hypothesis context
  - 5.$P \wedge False \implies False$ by (2)

This example demonstrates how UseFor produces lemmas that apply broadly and compress multiple reasoning steps into a single inference step in practice. This ability provides a crucial advantage in Monte Carlo Tree Search, where the search space expands exponentially with depth.

We remark that these proven conjectures are also observed to be very important to the prover in future iterations. For instance, $P \implies \neg\neg P$ and $1^{-1} = 1$ often serve as powerful shortcuts, condensing multi-step reasoning into a single step and thereby streamlining longer proofs.

Arithmetic	Propositional Logic	Group Theory
$\forall x \in \mathbb{N}, x(x^2+1) = x + x^3$	$False \iff (P \wedge False)$	$1^{-1} = 1$
$2x = 0 \implies x = 0$	$P \implies \neg\neg P$	
$\forall x \in \mathbb{N}, x * 1 = x$	$P \iff P$	$\forall x \in G, x \cdot x = x \implies x = 1$

Table 1: Representative conjectures judged extrinsically useful across three considered domains.

E Results for other domains

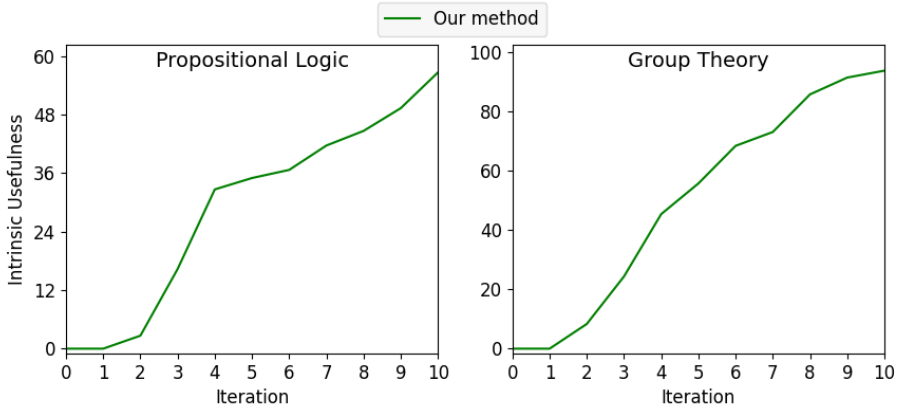

Figure 5: Intrinsic Evaluation: number of times lemmas were used during proving.

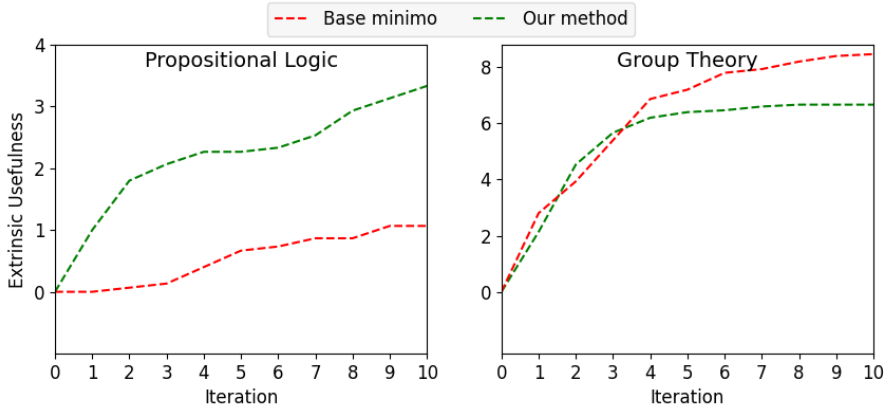

Figure 6: Extrinsic Evaluation: Number of deduplicated useful theorems per iteration, as determined by GPT-4.1 as a judge and proved by an SMT solver.