SELF-SUPERVISED DISCOVERY OF USEFUL THEOREMS IN FORMAL AXIOMATIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Artificial intelligence (AI) systems have recently advanced significantly in mathematical reasoning. Many approaches, including large language models (LLMs), use human prior knowledge from mathematical text, code, or theorem libraries. Although these approaches are highly effective in practice, it remains an open question whether an agent can autonomously discover useful theorems without such human priors. To examine this question, we develop an agent for discovering useful theorems from axioms alone. Our self-supervised theorem-discovery agent alternates between proof search and useful-theorem extraction, building a library of theorems reused as lemmas for further search. The agent discovers tens of thousands of theorems across three Hilbert-style propositional axiom systems, and its classical-logic instance also proves human-written benchmark problems. Providing the discovered theorems as prompt lemmas improves proof performance across multiple external LLMs on human-written and synthetic benchmarks. The benefits are larger on harder problems and persist in our evaluation with GPT-6 Astra. Qualitative examples show that both well-known and less obvious theorems contribute to proof construction. These results show that theorem discovery without human-provided theorem or problem data can reveal useful lemmas that humans did not explicitly provide.

1 INTRODUCTION

Artificial intelligence (AI) systems are rapidly improving their mathematical reasoning abilities. Large language models (LLMs) such as GPT, Claude, and Gemini have shown steady gains on mathematical benchmarks (OpenAI, 2024b; 2025; Anthropic, 2026; Gemini Team, 2025). More recently, OpenAI reported that a multi-agent system powered by an internal model produced an analytical proof addressing the Navier–Stokes existence and smoothness problem, one of the Millennium Prize Problems (OpenAI, 2026b).

Many current approaches combine pretrained language models with human-constructed mathematical content, including natural-language problem statements, formal proof corpora, and theorem libraries. Mathematical language models may be pretrained on mathematical text and code (Azerbayev et al., 2024), while autoformalization and proof-sketching methods connect human-written problem statements or informal proofs to formal proofs (Wu et al., 2022; Jiang et al., 2023). Proof-generation and retrieval-augmented methods often learn from or retrieve theorems and proofs in existing formal libraries (Han et al., 2022; Polu et al., 2023; Yang et al., 2023). Other methods scale formal proof search using synthetic or self-generated data (Xin et al., 2024; Zimmer et al., 2025; Hubert et al., 2026), or combine language-model generation with symbolic reasoning or learned theorem libraries (Trinh et al., 2024; Wang et al., 2024; Dong & Ma, 2025). By producing formally verifiable proofs, these approaches address a limitation of free-form LLM reasoning: its outputs can be difficult to verify and may contain plausible but incorrect statements (Farquhar et al., 2024; Kalai et al., 2025). Although highly effective, much of this literature still relies on human-provided mathematical content, making it difficult to isolate how much useful formal knowledge an agent can discover from axioms and inference rules alone.

To isolate what agents can learn from formal rules alone, a related line reduces reliance on human-provided proofs and existing theorem libraries by learning through exploration in axiomatized domains (Wu et al., 2021; Laurent & Platzer, 2022; Poesia et al., 2024). These studies show that agents

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

can self-improve and discover theorems or proofs in such domains. The closest prior work is that of Kasriel et al. (2025), but their approach differs from the setting considered here in two important respects. (i) They directly optimize a downstream usefulness objective based on theorem reuse and improvements in proof likelihood. (ii) They assess external usefulness with an LLM judge rather than measuring whether the discovered theorems improve the actual proof performance of an external reasoning agent. Consequently, it remains open whether useful theorems can emerge without directly optimizing downstream usefulness and measurably improve the proof performance of an independent reasoning agent.

In this study, we address this question in the controlled setting of Hilbert-style propositional logic, without an existing theorem library, a proof corpus, natural-language priors, or an externally provided training problem set. We formulate proof search as a stack-machine decision process whose actions push axioms or apply the inference rule. Our self-supervised algorithm reuses discovered theorems as future goals and trains a goal-conditioned policy on their proof action sequences. The algorithm selects useful theorems for reuse as actions in subsequent generations of proof search.

Experiments show that our agent discovers theorems that are useful both for the agent's own proof search and for proof search by external LLMs. Specifically, the agent discovers tens of thousands of theorems across three propositional axiom systems and finds proofs for human-written benchmark problems in the classical-logic setting. Moreover, when provided as prompt lemmas, the extracted theorems improve the proof performance of multiple LLMs on both the human-written benchmark and a broader synthetic benchmark. The benefits are larger on harder problems and persist in our evaluation with GPT-6 Astra. These results show that, within Hilbert-style propositional logic, useful theorems can emerge from proof search based only on axioms and inference rules, and that they can serve not only as internal lemmas for the agent but also as knowledge that assists external reasoning systems.

Contributions. The central finding of this study is that theorems can be discovered autonomously using only axioms and inference rules, and that these theorems can be useful to other reasoning agents. We support this finding through three contributions. First, we introduce a stack-machine formulation and a self-supervised method for discovering and reusing theorems from axioms alone in Sections 3 and 4. Second, we demonstrate autonomous theorem discovery and internal theorem reuse in Section 5.1. Third, we evaluate the discovered theorems as external knowledge in Section 5.2 and show that they improve the actual proof performance of LLMs not involved in theorem discovery.

2 PRELIMINARIES

In this section, we introduce the axioms and inference rule of the Hilbert system and our notation for propositional formulas. These definitions form the basis for the stack-machine decision process in Section 3.

Rules of the Hilbert System. A Hilbert system is an axiomatic proof system in which theorems are derived from axioms using inference rules (Hilbert & Ackermann, 1950; Mendelson, 2015). The system used in this paper, shown in Figure 1, consists of three axioms, `Ax1`, `Ax2`, `Ax3`, and one inference rule, Modus Ponens. The symbols A, B, C in the axioms stand for arbitrary formulas. A proof repeatedly introduces axioms and applies Modus Ponens, which derives Y from previously obtained formulas X and $X \to Y$. When the final formula matches the target, it has been derived as a theorem. The middle panel of Figure 1 shows a proof tree for the theorem $A \to A$.

Notation for Propositional Formulas. We take implication $\to$ and falsity $\bot$ as primitive. In classical propositional logic, the other connectives are syntactic sugar (Post, 1921). We define negation by $\neg A := A \to \bot$, disjunction by $A \lor B := (A \to \bot) \to B$, conjunction by $A \land B := (A \to (B \to \bot)) \to \bot$, and biconditional by $A \leftrightarrow B := (A \to B) \land (B \to A)$.

Soundness, Completeness, and Proof Search. This Hilbert system is sound and complete for classical propositional logic (Church, 1996). Nevertheless, deciding propositional validity is coNP-complete (Cook, 1971), and proof length can vary substantially across proof systems (Cook & Reckhow, 1979), making efficient search for short Hilbert proofs nontrivial.

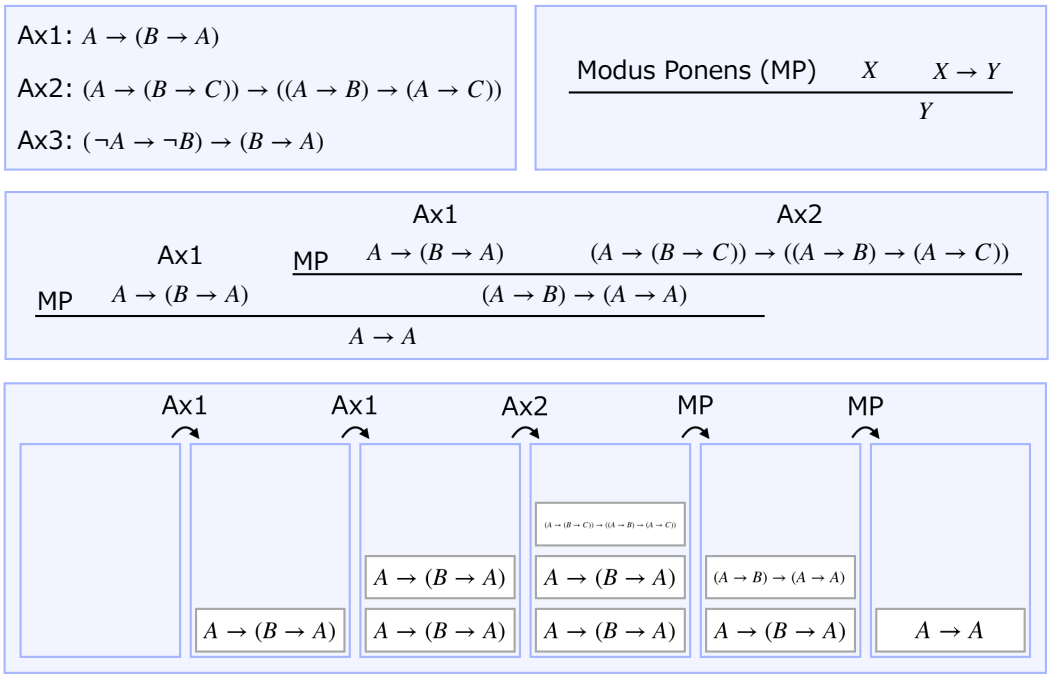


Figure 1: Hilbert-system proofs as stack-machine executions. Top: `Ax1`–`Ax3` and Modus Ponens. The axioms represent weakening, distribution of implication, and contraposition, respectively. Middle: a proof tree for $A \to A$. Bottom: the corresponding action sequence `Ax1`, `Ax1`, `Ax2`, `MP`, `MP`, where axiom actions push formulas onto the stack and `MP` combines the top two formulas.

3 Stack-Machine Formulation of Theorem Proving

We formulate proof construction in the Hilbert system as an action sequence executed by a stack machine. A Hilbert proof is represented as a proof tree built from axioms and Modus Ponens, as illustrated in the middle panel of Figure 1. For a learning algorithm, however, it is more natural to represent proof construction as a sequential decision process. This view is motivated by the Curry–Howard correspondence, which relates propositions to types and proofs to programs (Howard, 1980; Sørensen & Urzyczyn, 2006). In this analogy, implication resembles a function type and `MP` resembles function application. A postorder traversal maps any Hilbert proof tree to an action sequence in reverse Polish notation. Conversely, any successful stack-machine action sequence reconstructs a proof tree. Thus, the stack-machine representation preserves the set of provable formulas.

Formally, for a target formula g, we define the stack-machine decision process as a deterministic, goal-conditioned decision process given by the tuple $(\mathcal{S}, \mathcal{A}, s_0, P)$. Here, $\mathcal{S}$ is the set of finite stacks of formulas, $\mathcal{A} = \{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$ is the action space, and the initial state s_0 is the empty stack. The transition rule P is deterministic. For an axiom action, P pushes the corresponding Hilbert axiom schema shown in the top panel of Figure 1 onto the stack. For the `MP` action, we use Hindley–Milner unification over formula metavariables (Hindley, 1969; Milner, 1978; Damas & Milner, 1982). If the top formula can be unified with an implication $X \to Y$ and the formula immediately below it can be unified with X, the `MP` action is legal and pops both formulas, pushing the resulting consequent Y. The process succeeds when the stack consists of the single formula g. We do not specify a scalar reward function in this formulation, because the learning signals in our method are constructed from self-discovered theorems rather than externally defined rewards.

The bottom panel of Figure 1 shows the corresponding stack-machine execution for the proof of $A \to A$. Starting from an empty stack, two `Ax1` actions and one `Ax2` action push the corresponding axiom schemas onto the stack. The first `MP` action combines the top two formulas to derive $(A \to B) \to (A \to A)$. The second combines this result with the remaining `Ax1` formula, leaving the single formula $A \to A$ on the stack. The agent thus chooses which axiom to introduce and when to apply Modus Ponens, constructing the proof through a sequence of stack-machine actions.

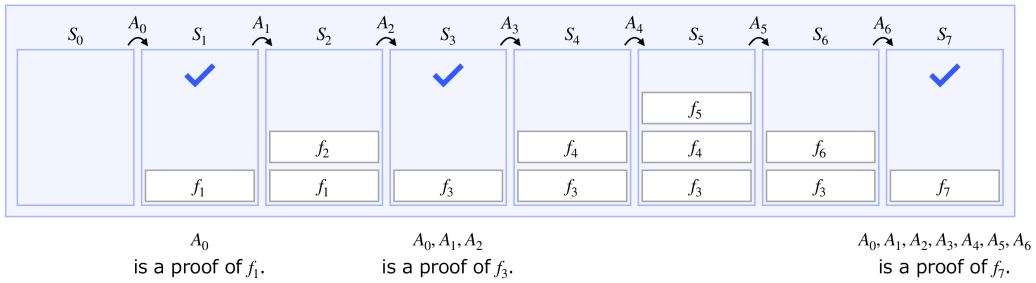


Figure 2: An example proof-search episode used to generate goals and training data. States marked with check marks are single-formula stacks, and the corresponding formulas f_1, f_3, and f_7 are added to the goal buffer $\mathcal{G}$. For each reached theorem, we use all preceding state-action prefixes as training examples: $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \left(\cup_{t\in\{1,3,7\}}\{(S_{t'}, A_{t'}, f_t) \mid 0 \le t' < t\}\right)$.

4 Discovering Useful Theorems from Axioms Alone

4.1 Learning a Goal-Conditioned Policy from Self-Discovered Theorems

Building on the stack-machine formulation in Section 3, we learn a goal-conditioned policy $\pi_\theta(A|S, g)$ from the agent's own proof-search trajectories. Here, S is the current proof stack, g is the target theorem, and A is the next stack-machine action.

During search, any state whose stack contains a single formula yields a reached theorem. If the state S_t contains only one formula f_t at time t, we call S_t a single-formula stack and treat f_t as a theorem reached in that episode, regardless of whether it matches the original goal formula g. Thus, even a failed attempt can provide a learning signal when a single-formula stack appears. Figure 2 illustrates this construction. The check-marked states S_1, S_3, and S_7 contain f_1, f_3, and f_7, and the corresponding action prefixes are proofs of those formulas. These reached theorems are then used in two ways, as future goals and as supervised training data.

We maintain a goal buffer $\mathcal{G}$ containing theorems discovered during search. Training begins by bootstrapping $\mathcal{G}$ with random rollouts. Whenever search produces new reached theorems, they are added to $\mathcal{G}$, and later iterations sample a goal formula g uniformly from $\mathcal{G}$ before rolling out the current policy $\pi_\theta(\cdot|S, g)$. Reached theorems also define supervised targets for the current policy update. When f_t is reached at time t, the prefix up to t is treated as a proof of f_t, and for each $0 \le t' < t$ we add $(S_{t'}, A_{t'}, f_t)$ to $\mathcal{D}_{\text{train}}$. The policy $\pi_\theta(A|S, g)$ is then updated by cross-entropy on $\mathcal{D}_{\text{train}}$, increasing the likelihood of $\pi_\theta(A_{t'}|S_{t'}, f_t)$. In Algorithm 1, the goal buffer $\mathcal{G}$ persists across iterations, whereas $\mathcal{D}_{\text{train}}$ is reset at the start of each iteration.

Algorithm 1 Policy Learning in a Fixed Action Space

Input: action space $\mathcal{A}$.
Initialize a new policy $\pi_\theta(A|S, g)$ for $\mathcal{A}$.
Bootstrap the goal buffer $\mathcal{G}$ with random rollouts.
for each iteration :
    Initialize current training set $\mathcal{D}_{\text{train}} \leftarrow \emptyset$.
    for each episode :
        Sample a goal formula g uniformly from $\mathcal{G}$.
        Roll out the current policy $\pi_\theta(\cdot|S, g)$ for at most T steps.
        for each time t at which S_t is a single-formula stack containing f_t :
            Add the reached theorem f_t to the goal buffer $\mathcal{G}$.
            Add the found proof path $\{(S_{t'}, A_{t'}, f_t) \mid 0 \le t' < t\}$ to $\mathcal{D}_{\text{train}}$.
        end for
    end for
    Update π_θ by minimizing cross-entropy loss on $\mathcal{D}_{\text{train}}$.
end for
return goal buffer $\mathcal{G}$.

Our learning rule relates to Hindsight Experience Replay and goal-conditioned supervised learning, which reuse reached states as goals (Andrychowicz et al., 2017; Ghosh et al., 2021). In our setting, the goals are proved theorems, some of which also become reusable actions in subsequent proof search.

4.2 Extracting Useful Theorems

We extract useful theorems from the discovered theorem set using two criteria, generality and reprovability. The goal buffer $\mathcal{G}$ constructed in the previous section serves as the candidate set. Generality removes overly specialized theorems, while reprovability prioritizes theorems that have already been discovered but cannot be reliably reproved by the current policy. Adding such theorems as lemmas can expand the set of theorems reached in subsequent proof search.

The first criterion uses the generality order induced by substituting formulas for variables to remove overly specialized theorems. For example, the theorem $A \to A$ is more general than the theorem $(A \to B) \to (A \to B)$, because substituting the formula $A \to B$ for A in the former yields the latter, whereas no variable substitution in the latter yields the former. We keep only the theorems that are not specializations of other candidates. Formally, we define a partial order so that more general theorems are smaller, and retain the minimal elements under this order. This filtering removes redundant specialized theorems while retaining more general candidates.

The second criterion ranks the remaining candidates by estimating how reliably the current policy can reprove them. For each candidate theorem $g \in \mathcal{G}$, we record the number n_g of times g was sampled as a proof goal and the number m_g of those episodes in which g was proved. A theorem with low reprovability has already been discovered, but the current policy cannot reliably prove it again. Making such a theorem available as a lemma can compress a difficult partial proof into a single lemma call. To estimate reprovability, we use the Bayesian posterior mean $\hat{p}_g = (m_g + 1)/(n_g + 2)$, which corresponds to a Beta$(1, 1)$ prior, rather than the raw empirical success rate. This smoothing prevents the ranking from over-prioritizing theorems that fail after only a few samples and helps identify theorems with persistently low reprovability. After generality filtering, we select a fixed number of candidates with the lowest reprovability posterior means $\hat{p}_g$.

4.3 Reusing Useful Theorems

The proposed method reuses extracted theorems by adding them to the action space of the stack-machine decision process. The original action space consists of four actions: pushing an instance of `Ax1`, `Ax2`, or `Ax3` onto the stack, and applying the inference rule `MP`. For each extracted theorem, we add one library action, denoted by `Thm1`, `Thm2`, ..., `ThmN`. When one of these theorem actions is selected, the corresponding theorem is pushed onto the stack instead of an axiom. The resulting stack-machine decision process allows the agent to reuse theorems discovered in earlier generations as lemmas in later proof search.

Algorithm 2 summarizes how the action space is expanded across generations. The algorithm alternates between policy learning under a fixed action space and action-space expansion using extracted theorems. In each generation, we train a new policy network for the current action space, because adding library actions changes the action space of the stack-machine decision process. During policy learning, reached theorems accumulate in the goal buffer $\mathcal{G}$ through multiple iterations of proof search and supervised updates. At the end of the generation, useful theorems are extracted from $\mathcal{G}$, and the corresponding library actions are added to the action space $\mathcal{A}$. This process turns theorems discovered in one generation into reusable action primitives for later generations. By repeating this process, the agent gradually expands its action space and grows a library of useful theorems.

Algorithm 2 Multi-Generation Library Expansion

1: Initialize action space $\mathcal{A} \leftarrow \{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$.
2: **for** each generation :
3: Give action space $\mathcal{A}$ to Algorithm 1 and obtain theorem buffer $\mathcal{G}$.
4: Extract useful theorems $\{\texttt{Thm1}, \ldots, \texttt{ThmN}\}$ from $\mathcal{G}$ based on generality and reprovability.
5: Expand action space $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{Thm1}, \ldots, \texttt{ThmN}\}$ for the next generation.
6: **end for**

Table 1: Axiom systems used in the theorem-discovery experiments.

Action	Name	Axiom	Classical	Positive impl.	Intuitionistic
Ax1	Weakening	$A \to (B \to A)$	✓	✓	✓
Ax2	Distribution	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	✓	✓	✓
Ax3	Contraposition	$(\neg A \to \neg B) \to (B \to A)$	✓		
Ax4	Explosion	$\neg A \to (A \to B)$			✓
Ax5	Absurdity	$(A \to B) \to ((A \to \neg B) \to \neg A)$			✓

Table 2: Theorem-selection and component ablations on the human-written benchmark derived from Kleene's textbook (Kleene, 1952).

(a) Theorem-selection baselines

Selection rule	Discovered theorems	Proof success rate
Random discovered	1.7×10^4	23%
Shortest discovered	2.0×10^4	17%
Most frequently reached	1.0×10^4	17%
Ours	$\mathbf{2.1 \times 10^4}$	**30%**

(b) Component ablations

Ablation	Discovered theorems	Proof success rate
No generality filtering	1.6×10^4	20%
No reprovability ranking	1.3×10^4	17%
No library-action reuse	8.2×10^1	17%
Full setting (ours)	$\mathbf{2.1 \times 10^4}$	**30%**

5 Experiments

5.1 Theorem Discovery and Internal Reuse

Setup. We first evaluate whether the proposed method can grow the goal buffer $\mathcal{G}$ starting only from the axioms and inference rule. To test applicability across axiom systems, we run the theorem-discovery procedure on classical logic, positive implicational logic, and intuitionistic logic. Table 1 summarizes the three axiom systems. Across axiom-system settings, we use the same training configuration and change only the initial action space in the first line of Algorithm 2 to match each axiom system. For each generation, we train a randomly initialized Transformer policy from scratch using only the agent's own proof-search trajectories. Appendix A provides architecture details.

Results. Figure 3 shows that the number of discovered theorems increases across generations in all three evaluated axiom systems. This growth is accompanied by substantial internal reuse: from the second generation onward, library actions are reused in 99.8% of successful proofs. Across seeds, the final theorem libraries have an average pairwise Jaccard index of 0.326, corresponding to approximately half of the theorems in each library being shared between runs. To assess whether the discovered theorems also include results meaningful beyond the agent's own search process, we evaluate the classical-logic agent on human-written benchmark problems derived from Kleene's textbook (Kleene, 1952) and curated by Poesia et al. (2024). The agent proves 30% of these problems, although none were provided as training targets. Together, these results show that the agent grows and actively reuses its theorem library while also discovering a non-trivial subset of human-written theorems.

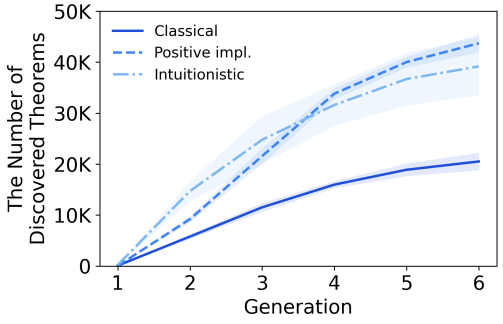
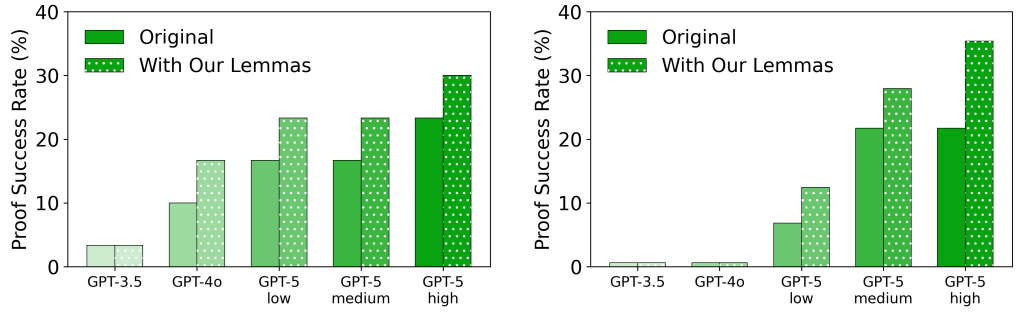


Figure 3: Number of discovered theorems across generations in the three evaluated axiom systems. The agent discovers tens of thousands of theorems in all three systems.

Theorem Selection and Component Ablations. We evaluate whether theorem selection based on generality and low reprovability is more effective than simpler selection rules and analyze the contribution of each component. As shown in Table 2(a), our method discovers approximately 2.1×10^4 theorems and achieves a proof success rate of 30% on the human-written benchmark derived from Kleene's textbook. Both metrics are the highest among the evaluated selection rules, indicating



(a) Human-written benchmark **(b)** Synthetic benchmark

Figure 4: Effect of the extracted theorems on external LLM proof success. When provided as prompt lemmas, the discovered theorems improve the proof performance of multiple LLMs on both the human-written and synthetic benchmarks.

that our selection rule effectively combines the discovery of a large number of theorems with the selection of theorems useful for human-written problems. Furthermore, as shown in Table 2(b), removing generality filtering, reprovability ranking, or library-action reuse reduces both the number of discovered theorems and the proof success rate. These results indicate that selection based on generality and low reprovability contributes to retaining useful theorems, while reusing them as library actions supports the continued growth of the theorem library.

5.2 Transfer to External LLMs

Setup. We evaluate whether the extracted theorems transfer as external knowledge for LLM proof search on two benchmarks. The first comprises 30 human-written propositional-logic problems derived from Kleene's textbook (Kleene, 1952) and curated by Poesia et al. (2024). The second is an independently constructed synthetic benchmark of 161 classically valid formulas over implication and falsity, containing at most three distinct propositional variables and six implication symbols. We exclude syntactically redundant formulas, overlaps with the 30-problem benchmark, and formulas matching the extracted theorems under variable renaming or substitution. We evaluate five GPT configurations: GPT-3.5, GPT-4o, and GPT-5 with low, medium, and high reasoning efforts (OpenAI, 2023; 2024a; 2025). For each model and benchmark, we compare proof success without prompt lemmas and with our extracted theorems provided as prompt lemmas. The evaluated LLMs do not participate in theorem discovery, and every generated proof is checked using the same deterministic proof verifier.

Results. Figure 4 compares proof success rates with and without the extracted theorems on the human-written and synthetic benchmarks. On the human-written benchmark, providing the extracted theorems improves proof success for GPT-4o and all three GPT-5 configurations. On the synthetic benchmark, all three GPT-5 configurations also improve. In particular, GPT-5-high solves 57 problems with the extracted theorems, compared with 35 without them. This corresponds to 22 additional solved problems and an increase in proof success rate from 21.7% to 35.4%, or 13.7 percentage points. In contrast, GPT-3.5 does not improve on either benchmark, and GPT-4o does not improve on the synthetic benchmark. These results provide evidence that the extracted theorems can serve as useful external knowledge for LLM proof search across different problem sets, particularly for LLMs with stronger reasoning capabilities.

Control Experiments with Random Valid Lemmas. To test whether the gains in Figure 4 arise merely from providing additional valid formulas, we compare our extracted theorems with two control lemma sets. We use random valid lemmas and complexity-matched random valid lemmas. Their construction is detailed in Appendix A. As shown in Table 3, GPT-5-high achieves its highest proof success rate with our lemmas on both the human-written and synthetic benchmarks. These controls suggest that the improvement cannot be explained merely by adding valid formulas or matching their surface complexity, supporting the importance of selecting lemmas useful for LLM proof search.

Table 3: GPT-5-high proof success with control lemmas. Across both benchmarks, the highest proof success rate is obtained with our lemmas.

Prompt lemmas	Human-written	Synthetic
None	23%	22%
Random valid	17%	20%
Complexity-matched	23%	25%
Our lemmas	**30%**	**35%**

Table 4: GPT-5-high proof success by problem difficulty. The extracted theorems improve both the number of solved problems and proof success rate, with larger gains for the hard group.

Problem group	Without lemmas	With our lemmas	Improvement
Easy (proof length ≤ 20)	25/38 (65.8%)	27/38 (71.1%)	+2 problems (+5.3 pt)
Hard (proof length > 20)	10/123 (8.1%)	30/123 (24.4%)	**+20 problems (+16.3 pt)**

Difficulty Analysis. To examine how the benefit of the extracted theorems varies with problem difficulty, we use the availability of a proof within 20 steps using only the four primitive actions $\{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$ as a difficulty proxy. Exhaustive search partitions the synthetic benchmark into 38 easy problems with such a proof and 123 hard problems not solved within this limit. As shown in Table 4, the extracted theorems enable GPT-5-high to solve two additional easy problems but 20 additional hard problems, corresponding to gains of 5.3 and 16.3 percentage points, respectively. The substantially larger gain on the hard group indicates that the extracted theorems are particularly useful when short proofs using only primitive actions are unavailable.

Qualitative Analysis of Extracted Theorems. To examine how the extracted theorems assist LLM proof construction, we analyze the following human-written benchmark problem.

$$g : A \to (\neg A \to B).$$

GPT-5-high failed to prove this goal without prompt lemmas but successfully proved it when given our extracted theorems. The resulting proof uses the following two extracted theorems as lemmas.

$$\texttt{Thm1} : \neg A \to (A \to B),$$
$$\texttt{Thm2} : (A \to (B \to C)) \to (D \to ((A \to B) \to (A \to C))).$$

GPT-5-high generates the following verified action sequence.

`Ax1, Thm1, Thm2, MP, Ax2, MP, MP.`

The two lemmas play different roles. `Thm1` expresses the principle of explosion, deriving an arbitrary B from $\neg A$ and A, and its usefulness is readily interpretable. In contrast, the usefulness of `Thm2` is less obvious from its statement alone. Nevertheless, it lifts an implication under an additional context and, together with `Thm1`, enables the proof of g. This example shows that the method can recover familiar, interpretable lemmas while also identifying less obvious theorems that provide useful guidance to an external LLM.

Evaluation with GPT-6 Astra. GPT-6 Astra is reported to exhibit strong mathematical reasoning performance (OpenAI, 2026a). To evaluate whether the extracted theorems remain useful with a more capable base model, we conduct additional experiments with GPT-6 Astra. In addition to the Human Written, Easy, and Hard problem sets, we construct an Extreme Hard benchmark to provide a sufficiently challenging evaluation and examine whether the benefit of the extracted theorems persists on harder problems. This benchmark consists of 100 randomly generated, classically valid goal formulas, each containing at least ten implication symbols, not solved within 20 primitive steps by exhaustive search using $\{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$, and containing no proper subformula that is itself valid or unsatisfiable. As shown in Table 5, providing the extracted theorems as prompt lemmas improves proof success on the Human Written,

Table 5: GPT-6 Astra proof success with and without our extracted theorems. The largest gain occurs on the Extreme Hard problem set.

Problem set	Without lemmas	With our lemmas	Improvement
Human Written	90%	93%	+3 pt
Easy	100%	100%	$\pm$0 pt
Hard	96%	97%	+1 pt
Extreme Hard	54%	75%	**+21 pt**

Hard, and Extreme Hard sets, while performance on Easy remains at 100%. The largest gain occurs on Extreme Hard, where proof success increases from 54% to 75%, an improvement of 21 percentage points. On the other problem sets, success without lemmas is already between 90% and 100%, leaving limited room for improvement. These results suggest that the extracted theorems remain useful for a highly capable LLM and provide the greatest benefit on problems not already saturated by the base model.

6 Related Work

Learning-guided formal theorem proving. Prior work on learning-guided theorem proving has studied reinforcement-learning proof search, tactic prediction, proof generation, premise retrieval, and LLM-based Lean proving (Kaliszyk et al., 2018; Han et al., 2022; Polu et al., 2023; Yang et al., 2023; Xin et al., 2024; Zimmer et al., 2025; Hubert et al., 2026). These methods build strong provers using existing formal libraries or proof data. In contrast, we ask how far theorem discovery and reuse can go using only the axioms and inference rules of a Hilbert system.

Self-play and intrinsic theorem discovery. Another line reduces dependence on human proof examples by learning proof-search or conjecturing through self-play or intrinsic motivation (Wu et al., 2021; Lample et al., 2022; Laurent & Platzer, 2022; Dong & Ma, 2025; Zhao et al., 2025). Poesia et al. (2024) discovers theorems from axioms but does not carry them forward as a persistent library for subsequent proof search. While Kasriel et al. (2025) is especially close, it trains a conjecturer and prover from axioms using a downstream usefulness signal based on theorem reuse and improvements in proof likelihood. Our method instead selects theorems reached during goal-conditioned proof search using generality and low reprovability, without directly optimizing downstream usefulness, and reuses them as lemmas across generations. Kasriel et al. assess external usefulness with an LLM judge, whereas we measure actual improvements in the proof performance of external LLMs not involved in discovery. These differences let us test whether useful theorems can emerge without such optimization and benefit a different reasoning agent.

Lemma discovery and library learning. Premise selection, lemma discovery, and library learning are important directions for accelerating proof search (Yang et al., 2023). Wang et al. (2024) grow a lemma library from externally provided problems using an LLM trained on human-generated data. In contrast to methods that rely on existing formal libraries, human-provided lemmas, or LLM-based proving pipelines, our agent discovers useful theorems through proof search using only the axioms and inference rule.

7 Conclusions

We formulated Hilbert-style propositional theorem proving as a stack-machine decision process and developed a self-supervised algorithm that builds a persistent theorem library from the agent's own proof search. Without human-provided theorem or problem data, the agent discovered tens of thousands of theorems across three propositional axiom systems. Ablations showed that generality, low reprovability, and persistent theorem reuse contributed to library growth and proof success. The discovered theorems aided the agent's own proof search and improved the actual proof performance of external LLMs on both human-written and broader synthetic benchmarks. A difficulty analysis found larger gains on harder problems for GPT-5-high, while qualitative examples showed how both interpretable and less obvious discovered theorems contributed to successful proofs. GPT-6 Astra also benefited from the extracted theorems, especially on the most challenging problems. These results indicate that axioms-only theorem discovery can produce reusable formal knowledge.

Limitations and Future Directions. To examine whether useful theorems can be discovered from axioms alone in a controlled setting, this study focuses on Hilbert-style propositional logic. Future work should investigate whether axioms-only theorem discovery can scale to richer formal systems, including first-order logic and set theory. A key challenge in extending our finite action-space formulation to first-order logic is handling quantifier instantiation, which can introduce an unbounded set of candidate terms. We hope our findings will motivate progress toward AI systems that autonomously build formal mathematical knowledge from axioms and inference rules.

AI USE STATEMENT

LLMs were evaluated as proof-generation systems, as described in Section 5 and Appendices A and C. The authors also used ChatGPT to improve the clarity and wording of the manuscript and to assist with experimental work. The authors reviewed the AI-assisted outputs and accept full responsibility for the manuscript, its scientific claims, and the accompanying research artifacts.

REPRODUCIBILITY STATEMENT

Appendix A describes the theorem-discovery training configuration, including the policy architecture and training hyperparameters, as well as the proof-verification protocol, control construction, and computational resources. Section 5.2 describes the construction of the 161-problem synthetic benchmark, including formula enumeration and filtering. Appendix C provides the prompt template used for LLM evaluation. The supplementary material includes code for reproducing the experiments.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5048–5058, 2017.

Anthropic. Claude Opus 4.7 system card. Technical report, Anthropic, April 2026. URL `https://www.anthropic.com/claude-opus-4-7-system-card`.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *International Conference on Learning Representations*, 2024.

Alonzo Church. *Introduction to Mathematical Logic*. Princeton University Press, Princeton, NJ, 1996. Reprint of the revised 1956 edition. See Propositions 235 and 239, pp. 127–128.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158. Association for Computing Machinery, 1971.

Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

Luís Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 207–212. Association for Computing Machinery, 1982.

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 14114–14136. PMLR, 2025.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024.

Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

David Hilbert and Wilhelm Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, New York, 1950. Translated by Lewis M. Hammond, George G. Leckie, and F. Steinhardt.

Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press, New York, 1980.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklos Z. Horvath, Goran Zuzic, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sonne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothée Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023.

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate. Technical report, 2025.

Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Timothe Kasriel, Thomas Lu, Devon Ding, Jingxuan He, and Dawn Song. Usefulness-driven learning of formal mathematics. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS*, 2025. MATH-AI 2025.

Stephen Cole Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Jonathan Laurent and André Platzer. Learning to find proofs and theorems by learning to refine search strategies: The case of loop invariant synthesis. In *Advances in Neural Information Processing Systems*, volume 35, pp. 4843–4856, 2022.

Elliott Mendelson. *Introduction to Mathematical Logic*. CRC Press, Boca Raton, 6 edition, 2015. ISBN 9781482237726.

Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

OpenAI. Introducing ChatGPT and Whisper APIs. OpenAI blog, 2023.

OpenAI. GPT-4o system card. System card, 2024a.

OpenAI. OpenAI o1 system card, 2024b.

OpenAI. GPT-5 system card. System card, 2025.

OpenAI. GPT-6 Astra system card. Technical report, OpenAI, September 2026a.

OpenAI. On the Navier–Stokes millennium prize problem. OpenAI Research, September 2026b. URL `https://openai.com/index/navier-stokes-solution/`.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *International Conference on Learning Representations*, 2023.

Emil L. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43(3):163–185, 1921.

Morten Heine Sørensen and Paweł Urzyczyn. *Lectures on the Curry–Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006. ISBN 9780444520777.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024.

Minchao Wu, Michael Norrish, Christian Walder, and Amir Dezfouli. TacticZero: Learning to prove theorems from scratch with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale synthetic data, 2024.

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. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Matthieu Zimmer, Xiaotong Ji, Rasul Tutunov, Anthony Bordg, Jun Wang, and Haitham Bou Ammar. Bourbaki: Self-generated and goal-conditioned MDPs for theorem proving, 2025.

A Experimental Details

In this section, we explain the experimental settings used for theorem-discovery training and library construction. For the experiments in Figure 3, we run the same theorem-discovery procedure in three Hilbert-style systems. Positive implicational logic uses `Ax1` and `Ax2`. Intuitionistic logic uses `Ax1`, `Ax2`, **`Ax4`** : $\neg A \to (A \to B)$, and **`Ax5`** : $(A \to B) \to ((A \to \neg B) \to \neg A)$. Classical logic uses `Ax1`, `Ax2`, and `Ax3` as shown in Figure 1. Let $\mathcal{T}_{\mathrm{pos}}$, $\mathcal{T}_{\mathrm{int}}$, and $\mathcal{T}_{\mathrm{cl}}$ denote the sets of theorems provable in positive implicational, intuitionistic, and classical logic, respectively. These theorem sets satisfy $\mathcal{T}_{\mathrm{pos}} \subsetneq \mathcal{T}_{\mathrm{int}} \subsetneq \mathcal{T}_{\mathrm{cl}}$.

We repeat proof search and library extraction over six generations. The numbers of extracted theorems added to the library are $20, 10, 5, 2, 1, 0$ from Generation 1 to Generation 6, which gives a final library size of 38. Each generation uses 25 training iterations. In each iteration, the agent generates 8192 episodes, and each episode has a maximum length of 7 steps. Therefore, each generation uses $25 \times 8192 \times 7$ simulator evaluations. Episodes are processed in batches of 2048 episodes.

The policy is parameterized by a Transformer network. The model uses embedding dimension 128, four Transformer blocks, four attention heads, feed-forward dimension 512, dropout rate 0.1, and learned positional encodings. The policy is trained with cross-entropy loss under a legal-action mask. We use Adam with learning rate 0.001, batch size 512, and one training epoch per iteration.

The policy observation is a fixed-length sequence of 1024 integer tokens. It consists of the sampled goal formula followed by the current proof stack, separated by a special field-separator token, and padded to a fixed length. Each formula is tokenized at the symbol level in reverse Polish notation, with tokens for implication, falsity, variables, and formula separators. Thus, a training example for $\pi_\theta(A|S, g)$ presents the goal formula g and the current stack S to the Transformer.

Across generations, the action space consists of the axiom actions for the current Hilbert-style system, the `MP` action, and the library theorem actions available at that generation. The legal-action mask is computed from the current stack, the remaining horizon, and the current action space. It is used to mask illegal actions during sampling and in the cross-entropy loss. When new theorems are extracted and added to the library, the action space changes accordingly. We therefore initialize and train a new policy network for the updated action space at each generation.

Evaluation protocol and proof verification. We evaluated proof success on the 30 human-written propositional-logic benchmark problems derived from Kleene's textbook (Kleene, 1952) and curated by Poesia et al. (2024). This benchmark provides a human-written test set that is independent of our theorem-discovery process. Each LLM baseline was evaluated once per problem with fixed decoding settings, so the reported success rates reflect single-shot proof generation rather than best-of-k sampling. A problem was counted as solved only when the generated action sequence was accepted by a stack-machine proof checker implementing `Ax1`–`Ax3`, theorem actions, and Modus Ponens with the same unification rule as in Section 3. In the lemma-augmented setting, each theorem action pushed the corresponding theorem from the extracted library onto the stack. Since each extracted theorem was stored with its primitive axiom/`MP` proof trajectory, lemma-augmented proofs can in principle be expanded into axiom-only Hilbert proofs.

Random-valid-lemma controls. For the controls in Table 3, we construct a theorem pool by exhaustively enumerating proof sequences of at most 20 primitive steps using the four primitive actions $\{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$. From this pool, we sample the same number of lemmas as used in our method. The random-valid control samples lemmas uniformly at random, whereas the complexity-matched control selects lemmas to match the formula lengths and syntax-tree depths of our extracted lemmas. Both controls use GPT-5-high and otherwise follow the same evaluation protocol as the corresponding experiment with our lemmas.

Compute Resources. The main experiments reported in this paper used approximately 50 GPU hours on NVIDIA GH200 GPUs with 120GB memory. Including preliminary experiments, ablations, and failed runs, we have spent roughly 5000 GPU hours on a mixture of NVIDIA A100 and NVIDIA GH200 GPUs. The supplementary code is configured to reproduce the main theorem-discovery pipeline.

B THEOREM RANKING

For Generation 1, the number of discovered theorems is small enough to enumerate in full, so Table 6 lists all theorems discovered in a representative single-seed run. Each row reports a discovered theorem in Formula f and an action sequence that proves it in Proof. Sampled denotes the number of times theorem f was sampled as a goal, written as n_f, and Solved denotes the number of successful reproofs, written as m_f. Ratio reports the Bayesian-smoothed reprovability estimate $\hat{p}_f = (m_f + 1)/(n_f+2)$ as a percentage, and the table is sorted in ascending order of this value. General indicates whether the theorem is minimal under the generality order, and Selected indicates whether it was added to the library used in Generation 2 and later, following the generality filter and ranking by Ratio.

Table 6: Theorem Ranking.

No.	Formula	Proof	Sampled	Solved	Ratio	General	Selected
1	$(A \to (B \to C)) \to (D \to ((A \to B) \to (A \to C)))$	Ax2 Ax1 Ax1 MP Ax2 MP MP	2443	0	0.04%	✓	✓
2	$A \to ((B \to (C \to D)) \to (B \to (C \to D)))$	Ax2 Ax1 Ax2 MP MP Ax1 MP	2435	0	0.04%		
3	$(\neg A \to \neg((B \to (C \to D)) \to ((B \to C) \to (B \to D)))) \to A$	Ax2 Ax1 MP Ax3 Ax2 MP MP	2416	0	0.04%	✓	✓
4	$((A \to B) \to A) \to ((A \to B) \to B)$	Ax1 Ax1 Ax2 MP MP Ax2 MP	2398	0	0.04%	✓	✓
5	$((A \to (B \to C)) \to (((A \to B) \to (A \to C)) \to D)) \to ((A \to (B \to C)) \to D)$	Ax2 Ax1 MP Ax2 Ax2 MP MP	2379	0	0.04%	✓	✓
6	$(A \to \neg\bot) \to (A \to A)$	Ax2 Ax3 Ax1 MP Ax2 MP MP	2362	0	0.04%		
7	$(\neg A \to \neg(B \to (C \to B))) \to A$	Ax1 Ax1 MP Ax3 Ax2 MP MP	2334	0	0.04%	✓	✓
8	$(\neg(A \to B) \to \neg C) \to ((C \to A) \to (C \to B))$	Ax3 Ax2 Ax1 MP Ax2 MP MP	2492	1	0.08%	✓	✓
9	$(\neg\bot \to \neg\bot) \to \neg\bot$	Ax3 Ax1 Ax2 MP Ax2 MP MP	2441	1	0.08%		
10	$(\neg A \to \neg B) \to (C \to (B \to A))$	Ax3 Ax1 Ax1 MP Ax2 MP MP	2370	1	0.08%	✓	✓
11	$\neg A \to (A \to B)$	Ax1 Ax3 Ax1 MP Ax2 MP MP	2449	2	0.12%	✓	✓
12	$(A \to B) \to ((C \to A) \to (C \to B))$	Ax1 Ax2 Ax1 MP Ax2 MP MP	2613	3	0.15%	✓	✓
13	$(A \to B) \to (A \to B)$	Ax1 Ax2 MP Ax1 Ax2 MP MP	2399	3	0.17%		
14	$(A \to (B \to C)) \to (A \to (B \to C))$	Ax2 Ax1 Ax2 MP MP	2357	4	0.21%		
15	$A \to (B \to (C \to A))$	Ax1 Ax1 Ax1 MP Ax2 MP MP	2343	4	0.21%	✓	✓
16	$(\neg\neg A \to \neg\neg B) \to (A \to B)$	Ax3 Ax3 Ax1 MP Ax2 MP MP	2313	4	0.22%	✓	✓
17	$((A \to (B \to C)) \to A) \to ((A \to (B \to C)) \to (B \to C))$	Ax2 Ax1 Ax2 MP MP Ax2 MP	2375	5	0.25%		
18	$((\neg A \to \neg B) \to B) \to ((\neg A \to \neg B) \to B)$	Ax3 Ax2 MP Ax1 Ax2 MP MP	2361	7	0.34%		
19	$((A \to (B \to C)) \to (A \to B)) \to ((A \to (B \to C)) \to (A \to B))$	Ax2 Ax2 MP Ax1 Ax2 MP MP	2409	17	0.75%		
20	$((\neg A \to \neg B) \to ((B \to A) \to C)) \to ((\neg A \to \neg B) \to C)$	Ax3 Ax1 MP Ax2 Ax2 MP MP	2377	283	11.94%	✓	✓
21	$(\neg A \to \neg(\neg A \to \neg A)) \to A$	Ax1 Ax2 MP Ax3 Ax2 MP MP	2395	682	28.49%	✓	✓
22	$(A \to (B \to C)) \to (((A \to B) \to A) \to ((A \to B) \to C))$	Ax2 Ax2 Ax1 MP Ax2 MP MP	2381	1051	44.15%	✓	✓
23	$A \to ((\neg\bot \to \neg\bot) \to (\neg\bot \to \neg\bot))$	Ax3 Ax2 MP Ax3 MP Ax1 MP	2337	1312	56.14%		
24	$(A \to ((B \to A) \to C)) \to (A \to C)$	Ax1 Ax1 MP Ax2 Ax2 MP MP	2493	1607	64.45%	✓	✓
25	$A \to (B \to B)$	Ax1 Ax1 Ax2 MP MP Ax1 MP	2429	1635	67.30%	✓	✓
26	$A \to \neg\bot$	Ax1 Ax2 MP Ax3 MP Ax1 MP	2407	1655	68.74%		
27	$(\bot \to A) \to (\bot \to A)$	Ax1 Ax2 MP Ax2 MP Ax3 MP	2310	1613	69.81%		
28	$((\neg A \to \neg B) \to \neg A) \to ((\neg A \to \neg B) \to \neg B)$	Ax3 Ax1 Ax2 MP MP Ax2 MP	2369	1666	70.31%		
29	$(A \to (A \to B)) \to (A \to B)$	Ax1 Ax2 MP Ax2 Ax2 MP MP	2377	1697	71.37%	✓	✓
30	$((\neg\bot \to \neg\bot) \to \neg\bot) \to ((\neg\bot \to \neg\bot) \to \neg\bot)$	Ax3 Ax2 MP Ax3 MP Ax2 MP	2452	1794	73.15%		
31	$\neg\bot$	Ax1 Ax2 MP Ax3 MP	2384	1759	73.76%		
32	$A \to A$	Ax1 Ax1 Ax2 MP MP	2353	1810	76.90%	✓	✓
33	$A \to ((\neg B \to \neg C) \to (\neg B \to \neg C))$	Ax3 Ax1 Ax2 MP MP Ax1 MP	2362	1855	78.51%		
34	$(\neg A \to \neg((\neg B \to \neg C) \to (C \to B))) \to A$	Ax3 Ax1 MP Ax3 Ax2 MP MP	2423	1921	79.26%	✓	✓
35	$(\neg A \to \neg B) \to (\neg A \to \neg B)$	Ax3 Ax1 Ax2 MP MP	2416	1992	82.42%		
36	$((\neg A \to \neg(A \to B)) \to (A \to B)) \to ((\neg A \to \neg(A \to B)) \to B)$	Ax3 Ax2 MP Ax2 Ax2 MP MP	2406	2001	83.14%	✓	✓
37	$(\neg\bot \to \neg\bot) \to (\neg\bot \to \neg\bot)$	Ax3 Ax2 MP Ax3 MP	2375	1982	83.42%		
38	$(A \to (B \to C)) \to (A \to (B \to B))$	Ax1 Ax2 MP Ax1 MP Ax2 MP	2478	2112	85.20%	✓	
39	$((A \to B) \to A) \to ((A \to B) \to (C \to B))$	Ax1 Ax1 MP Ax2 MP Ax2 MP	2359	2026	85.85%	✓	
40	$A \to (((B \to C) \to B) \to ((B \to C) \to B))$	Ax1 Ax2 MP Ax2 MP Ax1 MP	2399	2061	85.88%		
41	$(((\neg A \to \neg B) \to B) \to (\neg A \to \neg B)) \to (((\neg A \to \neg B) \to B) \to A)$	Ax3 Ax2 MP Ax2 MP	2381	2048	85.98%	✓	
42	$A \to ((((\neg B \to \neg C) \to C) \to (\neg B \to \neg C)) \to (((\neg B \to \neg C) \to C) \to B))$	Ax3 Ax2 MP Ax2 MP Ax1 MP	2488	2147	86.27%	✓	

Continued on next page

Table 6: Theorem Ranking, continued.

No.	Formula	Proof	Sampled	Solved	Ratio	General	Selected
43	$((A \to (\neg B \to \neg C)) \to A) \to ((A \to (\neg B \to \neg C)) \to (C \to B))$	Ax3 Ax1 MP Ax2 MP Ax2 MP	2404	2077	86.37%		✓
44	$A \to ((B \to C) \to (B \to (D \to C)))$	Ax1 Ax1 MP Ax2 MP Ax1 MP	2405	2080	86.46%		✓
45	$(((A \to B) \to A) \to (A \to B)) \to (((A \to B) \to A) \to A)$	Ax1 Ax2 MP Ax2 MP Ax2 MP	2424	2135	88.05%		
46	$A \to ((B \to (\neg C \to \neg D)) \to (B \to (D \to C)))$	Ax3 Ax1 MP Ax2 MP Ax1 MP	2359	2078	88.06%		✓
47	$(A \to (\neg B \to \neg C)) \to (A \to (C \to B))$	Ax3 Ax1 MP Ax2 MP	2397	2113	88.12%		✓
48	$(A \to B) \to (A \to ((C \to (D \to E)) \to ((C \to D) \to (C \to E))))$	Ax2 Ax1 MP Ax1 MP Ax2 MP	2302	2030	88.15%		
49	$A \to ((B \to C) \to (B \to B))$	Ax1 Ax2 MP Ax1 MP	2393	2118	88.48%		✓
50	$A \to (B \to ((C \to D) \to (C \to C)))$	Ax1 Ax2 MP Ax1 MP Ax1 MP	2349	2082	88.60%		✓
51	$(A \to B) \to (A \to (C \to (D \to C)))$	Ax1 Ax1 MP Ax1 MP Ax2 MP	2484	2203	88.66%		
52	$(A \to B) \to (A \to (C \to B))$	Ax1 Ax1 MP Ax2 MP	2493	2224	89.18%		✓
53	$(A \to ((\neg B \to \neg C) \to C)) \to (A \to ((\neg B \to \neg C) \to B))$	Ax3 Ax2 MP Ax1 MP Ax2 MP	2339	2094	89.49%		✓
54	$((A \to B) \to A) \to ((A \to B) \to A)$	Ax1 Ax2 MP Ax2 MP	2380	2131	89.50%		
55	$(A \to (B \to (C \to D))) \to (A \to ((B \to C) \to (B \to D)))$	Ax2 Ax1 MP Ax2 MP	2324	2081	89.51%		✓
56	$A \to (B \to (((\neg C \to \neg D) \to D) \to ((\neg C \to \neg D) \to C)))$	Ax3 Ax2 MP Ax1 MP Ax1 MP	2353	2107	89.51%		✓
57	$(A \to ((B \to (C \to D)) \to (B \to C))) \to (A \to ((B \to (C \to D)) \to (B \to D)))$	Ax2 Ax2 MP Ax1 MP Ax2 MP	2434	2181	89.57%		✓
58	$A \to (B \to (C \to ((D \to (E \to F)) \to ((D \to E) \to (D \to F)))))$	Ax2 Ax1 MP Ax1 MP Ax1 MP	2453	2210	90.06%		✓
59	$((((\neg A \to \neg B) \to B) \to (\neg A \to \neg B)) \to ((\neg A \to \neg B) \to B)) \to ((((\neg A \to \neg B) \to B) \to (\neg A \to \neg B)) \to A)$	Ax3 Ax2 MP Ax2 MP Ax2 MP	2345	2118	90.29%		✓
60	$((A \to (B \to (C \to D))) \to A) \to ((A \to (B \to (C \to D))) \to ((B \to C) \to (B \to D)))$	Ax2 Ax1 MP Ax2 MP Ax2 MP	2335	2114	90.50%		✓
61	$A \to ((((B \to (C \to D)) \to (B \to C)) \to (B \to (C \to D))) \to (((B \to (C \to D)) \to (B \to C)) \to (B \to D)))$	Ax2 Ax2 MP Ax2 MP Ax1 MP	2496	2261	90.55%		✓
62	$A \to ((B \to (C \to (D \to E))) \to (B \to ((C \to D) \to (C \to E))))$	Ax2 Ax1 MP Ax2 MP Ax1 MP	2444	2214	90.56%		✓
63	$(((A \to (B \to C)) \to (A \to B)) \to (A \to (B \to C))) \to (((A \to (B \to C)) \to (A \to B)) \to (A \to C))$	Ax2 Ax2 MP Ax2 MP	2360	2146	90.90%		✓
64	$(A \to B) \to (A \to ((\neg C \to \neg D) \to (D \to C)))$	Ax3 Ax1 MP Ax1 MP Ax2 MP	2359	2146	90.94%		
65	$((((A \to (B \to C)) \to (A \to B)) \to (A \to (B \to C))) \to (((A \to (B \to C)) \to (A \to B)) \to (A \to C))) \to (((A \to (B \to C)) \to (A \to B)) \to (A \to (B \to C))) \to (A \to C))$	Ax2 Ax2 MP Ax2 MP Ax2 MP	2493	2287	91.70%		✓
66	$A \to (B \to (C \to ((\neg D \to \neg E) \to (E \to D))))$	Ax3 Ax1 MP Ax1 MP Ax1 MP	2438	2239	91.80%		✓
67	$A \to (B \to ((\neg C \to \neg D) \to (D \to C)))$	Ax3 Ax1 MP Ax1 MP	2422	2227	91.91%		✓
68	$A \to (B \to (((C \to (D \to E)) \to (C \to D)) \to ((C \to (D \to E)) \to (C \to E))))$	Ax2 Ax2 MP Ax1 MP Ax1 MP	2399	2208	92.00%		✓
69	$A \to (B \to (C \to (D \to (E \to D))))$	Ax1 Ax1 MP Ax1 MP Ax1 MP	2338	2153	92.05%		✓
70	$A \to (((B \to (C \to D)) \to (B \to C)) \to ((B \to (C \to D)) \to (B \to D)))$	Ax2 Ax2 MP Ax1 MP	2448	2258	92.20%		✓
71	$A \to (((\neg B \to \neg C) \to C) \to ((\neg B \to \neg C) \to B))$	Ax3 Ax2 MP Ax1 MP	2405	2227	92.56%		✓
72	$((\neg A \to \neg B) \to B) \to ((\neg A \to \neg B) \to A)$	Ax3 Ax2 MP	2322	2157	92.86%		✓
73	$A \to (B \to ((C \to (D \to E)) \to ((C \to D) \to (C \to E))))$	Ax2 Ax1 MP Ax1 MP	2392	2224	92.94%		✓
74	$(A \to B) \to (A \to A)$	Ax1 Ax2 MP	2433	2264	93.02%		
75	$A \to (B \to (C \to (D \to C)))$	Ax1 Ax1 MP Ax1 MP	2361	2213	93.69%		✓
76	$((A \to (B \to C)) \to (A \to B)) \to ((A \to (B \to C)) \to (A \to C))$	Ax2 Ax2 MP	2337	2225	95.17%		✓
77	$A \to ((B \to (C \to D)) \to ((B \to C) \to (B \to D)))$	Ax2 Ax1 MP	2419	2334	96.45%		✓
78	$A \to ((\neg B \to \neg C) \to (C \to B))$	Ax3 Ax1 MP	2238	2171	96.96%		✓
79	$A \to (B \to (C \to B))$	Ax1 Ax1 MP	2376	2306	97.01%		✓
80	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	Ax2	2413	2374	98.34%		✓
81	$A \to (B \to A)$	Ax1	2454	2421	98.62%		✓
82	$(\neg A \to \neg B) \to (B \to A)$	Ax3	2448	2437	99.51%		✓

C PROMPTS FOR LLM EVALUATION

We show the prompt template used for LLM evaluation. In the prompt, the axioms corresponding to `Ax1`, `Ax2`, and `Ax3` in the main text are denoted by 'A01', 'A02', and 'A03', respectively, and Modus Ponens is denoted by 'MP'. When discovered theorems are provided as lemmas, they are listed as 'L01', 'L02', and so on. For the baseline without lemmas, this lemma list is empty, and the LLM is asked to construct a proof using only 'A01', 'A02', 'A03', and 'MP'. The final answer is requested as an RPN sequence in which actions are separated by spaces.

```
# Rules
Prove the given proposition in the Hilbert system (with axioms A01, A02,
A03 and the inference rule Modus Ponens).
```

```
## Axioms:
A01: A -> (B -> A)
A02: (A -> (B -> C)) -> ((A -> B) -> (A -> C))
A03: (!A -> !B) -> (B -> A)

## Inference Rule (MP):
In the stack, the top element is regarded as X->Y, and the one below it
as X. Unify the antecedents of X and X->Y as in Hindley-Milner type
inference, and as a result derive the consequent Y, which is then pushed
onto the stack. For example, if the top of the stack is (A->(B->C))->((A
->B)->(A->C)) and the next one is A->(B->A), then A->(B->C) and A->(B->A)
 are unified with A=C, yielding the result (A->B)->(A->A).

## Lemmas:
You can use the following lemmas if you need.
L01: (A -> B) -> ((C -> A) -> (C -> B))
L02: (A -> (B -> C)) -> (B -> (A -> C))

# Output Format
The proof must ultimately be represented as a combinator expression in
Reverse Polish Notation (RPN). In RPN, A01, A02, A03 each push an
instance of the corresponding axiom onto the stack, L01, L02, ... do the
same for lemmas, and MP represents Modus Ponens according to the above
rule. You may include explanations, but enclose the final proof
expression between <final_answer> and </final_answer>.

# Example Problem
Target proposition: A -> A
Reference proof steps:
1. A -> (B -> A) (Axiom A01)
2. A -> (B -> A) (Axiom A01)
3. (A -> (B -> C)) -> ((A -> B) -> (A -> C)) (Axiom A02)
4. (A -> B) -> (A -> A) (Apply MP to 3 and 2)
5. A -> A (Apply MP to 4 and 1)
In this case, the combinator expression in RPN is A01 A01 A02 MP MP.
Final output example is the following:
<final_answer>A01 A01 A02 MP MP</final_answer>

# Problem
Prove the following proposition:
(A -> B) -> ((B -> C) -> (A -> C))
```

D Additional Examples of Extracted Theorems Helping LLM Proofs

In addition to the example in Section 5.2, we provide further examples in which extracted theorems help LLM proof search. In both goal theorems shown here, GPT-5-high failed to find a proof without lemmas, but succeeded when the extracted theorems were supplied as lemmas. These examples provide additional evidence that the extracted theorems not only appear useful, but also actually help proof construction in external LLM-based proof search.

The two corresponding goal theorems are shown below.

Table 7: Goal theorems in the additional examples.

Goal No.	Goal Formula
1	$(A \to B) \to ((B \to C) \to (A \to C))$
2	$\neg((A \to B) \to \neg(B \to A)) \to (B \to A)$

The prompt contained the extracted theorem library, and `Thm1` to `Thm5` denote the members of this library that were actually used by the LLM in the generated proofs.

Table 8: Extracted theorem actions utilized by GPT-5-high in the additional examples.

Action	Theorem Formula
`Thm1`	$\neg A \to (A \to B)$
`Thm2`	$(A \to (B \to C)) \to (D \to ((A \to B) \to (A \to C)))$
`Thm3`	$(A \to B) \to ((C \to A) \to (C \to B))$
`Thm4`	$(\neg A \to \neg(B \to (C \to B))) \to A$
`Thm5`	$((A \to (B \to C)) \to (((A \to B) \to (A \to C)) \to D)) \to ((A \to (B \to C)) \to D)$

The proof action sequences generated by GPT-5-high with lemmas are shown below. In each proof, `Ax1`, `Ax2`, and `Ax3` push the corresponding axioms onto the stack, `Thm1` to `Thm5` push the corresponding extracted theorems, and `MP` applies Modus Ponens.

Table 9: Proof action sequences generated by GPT-5-high with lemmas.

Goal No.	Proof
1	`Thm3 Ax1 Ax1 MP Thm3 Ax2 MP MP Ax1 MP Thm5 MP MP`
2	`Thm1 Ax1 Ax1 MP Ax1 Thm3 MP MP Thm3 Thm2 MP Ax2 MP MP Ax2 MP MP Thm4 Thm3 MP MP`

Table 9 makes the role of individual extracted theorems explicit. `Thm3` and `Thm5` are used to prove Goal 1, and `Thm1`, `Thm2`, `Thm3`, and `Thm4` are used to prove Goal 2. These results show that extracted theorems do not merely improve aggregate evaluation metrics, but also play concrete roles in individual proof constructions. In particular, not only human-interpretable theorems such as `Thm1` and `Thm3`, but also theorems whose usefulness is not obvious at first glance, such as `Thm2`, `Thm4`, and `Thm5`, function as effective supporting knowledge when an LLM constructs Hilbert-system proofs.

E Additional Evaluations of Efficiency and Robustness

Latency. Figure 5 reports the average latency over all proof attempts, including both successful and unsuccessful attempts. Providing the extracted theorems reduces average latency for GPT-5-medium and GPT-5-high. Because wall-clock latency depends on the model and serving environment, we treat this result as supplementary evidence rather than a central finding.

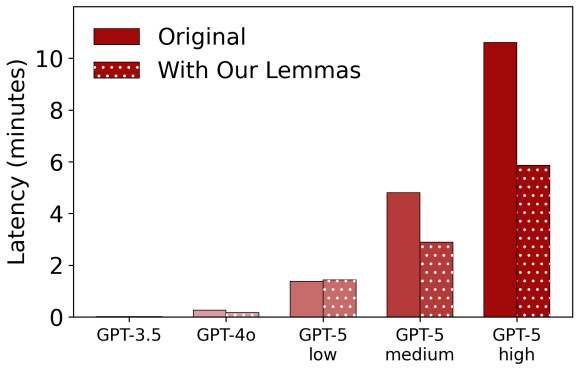


Figure 5: Average proof-search latency with and without our extracted theorems. Averages include both successful and unsuccessful attempts.

Best-of-5 Evaluation. We conduct a best-of-5 evaluation in which each model generates five candidate proofs for each problem, and a problem is counted as solved if at least one candidate is ac-

cepted by the proof checker. As shown in Table 10, providing the extracted theorems as prompt lemmas improves proof success for all evaluated LLMs except GPT-3.5. In particular, GPT-5-medium and GPT-5-high both achieve improvements of 10.0 percentage points. These results show that the benefits of the extracted theorems are not limited to single-shot evaluation and persist under a larger inference budget.

Table 10: Best-of-5 proof success rates on the human-written benchmark.

Model	Without lemmas	With our lemmas	Improvement
GPT-3.5	2/30 (6.7%)	2/30 (6.7%)	$\pm$0.0 pt
GPT-4o	3/30 (10.0%)	4/30 (13.3%)	+3.3 pt
GPT-5-low	5/30 (16.7%)	6/30 (20.0%)	+3.3 pt
GPT-5-medium	5/30 (16.7%)	8/30 (26.7%)	**+10.0 pt**
GPT-5-high	8/30 (26.7%)	**11/30 (36.7%)**	**+10.0 pt**

Evaluation with Tool Use. We allow GPT-5-high to use either a proof checker or a Python interpreter. The proof checker is identical to the one used for the final correctness evaluation. It executes a candidate proof sequence and returns the derived formula together with whether the proof succeeded. In the Python-interpreter setting, the model can freely define and execute functions. As shown in Table 11, the extracted theorems improve proof success in the prompt-only, proof-checker, and Python-interpreter settings. However, tool use itself does not improve performance over the prompt-only setting in these experiments. We do not claim that these tool interfaces are optimal. Rather, the results show that the extracted theorems can provide complementary knowledge even when the LLM has access to external computational tools.

Table 11: GPT-5-high proof success rates with tool use on the human-written benchmark.

Setting	Without lemmas	With our lemmas
Prompt only	7/30 (23.3%)	**9/30 (30.0%)**
With proof checker	7/30 (23.3%)	**8/30 (26.7%)**
With Python interpreter	5/30 (16.7%)	**8/30 (26.7%)**