Disassembling the LEGO-Prover: With In-Context LLM Library Learning, Compute is King

Anonymous ACL submission

Abstract

The in-context learning (ICL) coding, reasoning, and tool-using ability of LLMs has spurred interest in library learning (i.e., the creation and exploitation of reusable and composable functions, tools, or lemmas). Such systems often promise improved task performance and computational efficiency by caching reasoning (i.e., the storage of generated tools) – all without finetuning. However, we find strong reasons to be skeptical. We perform an in-depth examination of one such system, LEGO-Prover, which purports to learn reusable lemmas for mathematical reasoning. We find no evidence of the direct reuse of learned lemmas, and find evidence against the soft reuse of learned lemmas (i.e., reuse by modifying relevant examples). Crucially, we find that LEGO-Prover does *not* improve over the simple baseline of prompting the model – improvements in task accuracy vanish once computational cost is accounted for. Concerningly, we again find no improvement after cost correction in two other ICL library learning systems: TroVE and AgentOptimizer.

Our findings suggest that a serious re-examination of the effectiveness of ICL LLM-based library learning is required, as is much stronger standards for evaluation including behavioural analysis and ensuring an equal computational budget is used for baselines.[1]

1 Introduction

Humans extend their capabilities, improve their reliability, and save mental effort through creating broadly applicable, composable, and reusable tools and knowledge (Sweller, 1988). Whether software libraries, standard operating procedures, or mathematical lemmas, these artifacts allow us to save effort, act predictably, act accurately, and more easily solve tasks. Library learning systems aim to artificially reproduce this ability; e.g., DreamCoder (Ellis et al., 2021) can learn to write code for complex text editing and LOGO drawing tasks by building up a library of increasingly complex composable functions.

Recent advances in the coding, reasoning, and tool-use ability of LLMs has raised the possibility of performing library learning with LLMs. Coding ability is required to create reusable and composable code, and tool-use ability is needed to leverage these self-created tools. Indeed, within the last year, several works have leveraged in-context learning to perform library learning – prompting the LLM to generate and apply broadly applicable helper tools. The primary benefit claimed is typically improved performance (Nguyen et al., 2024), though the proposed reasons for these gains vary. Compared to directly generating task-specific code, Wang et al. (2024c) and Yuan et al. (2024) hypothesize that library learning encourages simpler and more reliable functions. Similarly, Qian et al. (2023) suggest that performance improves because the task is broken down into separate abstract reasoning and instantiation tasks (i.e., creating a generic tool before calling it with the specifics of the given problem). Wang et al. (2024a) propose that their system works through the accumulation of reusable relevant reference examples. These systems may also claim reduced costs through caching these created tools (Cai et al., 2024).

In this work, we investigate the internal behaviour and black box performance of one such system: LEGO-Prover (Wang et al., 2024a). LEGO-Prover (i.e., LP in tables) is a LLM-based library learning system built for the autoformalization of theorems, i.e., converting natural language proofs into formal theorems verifiable by the Isabelle theorem prover (Paulson, 1994). Specifically, Wang et al. (2024a) reported SotA performance at the time of publication, and stated that "LEGO-Prover enables LLMs to utilize existing skills retrieved from the library" and "[m]odular and reusable skills are constantly added to the library to enable tack-

[1] We will release our code on GitHub upon publication.

ling increasingly intricate mathematical problems." Based on these claims, we aim to answer two important questions. First: Does LEGO-Prover exhibit library learning behaviours? Second: Does LEGO-Prover improve performance?

Our contributions are as follows. (1) For a variety of LLMs, we demonstrate that LEGO-Prover's learned library is not directly reused. (2) For a variety of LLMs, we provide evidence that LEGO-Prover's learned library does not serve as a useful reference for solving related tasks. (3) We demonstrate that LEGO-Prover does not outperform repeatedly prompting the model after accounting for computational budget. (4) We demonstrate that two other ICL LLM library learning systems, TroVE (Wang et al., 2024d) and AgentOptimizer (Zhang et al., 2024), likewise fail to outperform repeated prompting under an equivalent computational budget. Together, these points suggests that a serious re-examination of ICL LLM-based library learning is required. (5) Based on our findings, we make concrete proposals for improving the development and evaluation of library learning systems.

2 Background

LEGO-Prover was designed to convert natural language proofs into verifiable formal proofs via two families of concurrent processes that interact via a shared database: the PROVER and EVOLVER.

The PROVER iterates over pending tasks: For each task, it first decomposes the natural language proof into requests for potentially useful lemmas (in the form of Isabelle lemma statements without proofs) and then rewords the natural language proof as a step-by-step proof, still in English. This step-by-step proof is then augmented with retrieved lemmas and converted into a formal proof which is either verified or rejected by Isabelle. Simultaneously, the EVOLVER attempts to prove the requested potentially useful lemmas, adding these to the shared database. The EVOLVER also attempts to "evolve" existing lemmas to make them more useful for the solving of retrieved pending tasks or lemma requests. The PROVER cycles through a pool of unsolved problems; the maximum number of attempts per problem is a hyperparameter. In both the PROVER and EVOLVER, all steps are performed via in-context learning with examples. Retrieval is performed via Ada embeddings.

We use Draft-Sketch-Prove (Jiang et al., 2023) as a baseline, as done by Wang et al. (2024a). Draft-Sketch-Prove (i.e., DSP in tables) uses in-context learning to directly convert a natural language proof into an Isabelle proof, applying post-processing with heuristics to fill or correct gaps in the proof. In our experiments we use LEGO-Prover's post-processor and wrapper of the Isabelle verifier, increasing the timeout to 120s as used by Draft-Sketch-Prove – this wrapper was derived from Draft-Sketch-Prove and applies a superset of the original Draft-Sketch-Prove heuristics for correcting generated proofs.

As the original implementation of Draft-Sketch-Prove generated all proof attempts before performing verification, we modify Draft-Sketch-Prove to alternate between proof generation and evaluation as LEGO-Prover does. I.e., once a problem is solved we cease generating proofs for this task.

3 Evaluating LEGO-Prover Under Various LLMs

We begin by evaluating LEGO-Prover and Draft-Sketch-Prove on the miniF2F (Zheng et al., 2021) test set using three different LLMs (see Table 1). Following Wang et al. (2024a) we run all models for a fixed number of prover attempts. We improve upon the original experimental procedure by performing three trials and reporting the standard deviation. We detail models and hyperparameters in Appendix A.

Note that our experiments on the full miniF2F test set use gpt-4o-mini as the underlying LLM instead of ChatGPT-3.5-Turbo as used by (Wang et al., 2024a). This model was selected due to its superior cost effectiveness and improved reasoning, coding, and mathematics proficiency (OpenAI, 2024). We find that our Draft-Sketch-Prove performance of $35.9 \pm 0.9\%$ using gpt-4o-mini is close to the 38.5% GPT-3.5-Turbo accuracy reported by Wang et al. (2024a). This makes the discrepancy with Wang et al. (2024a)'s reported 45.5% (or 50% using human proofs) LEGO-Prover accuracy all the more striking. We investigated whether LEGO-Prover may be particularly sensitive to prompt, finding no such evidence (see Appendix C).

With the exception of gpt-4o-mini, these results *appear* to confirm the findings of Wang et al. (2024a): LEGO-Prover outperforms Draft-Sketch-Prove which suggests a benefit from library learning (at the very least under more powerful models).

	gpt-4o-mini*	gpt-4o†	gpt-4o‡	gpt-4o§	o3-mini‡	o3-mini§
DSP	**35.9 ± 0.9%**	23.1 ± 1.2%	25.0 ± 0.0%	25.0 ± 0.0%	19.4 ± 2.4%	19.4 ± 4.8%
LP	25.8 ± 1.6%	**29.3 ± 3.1%**	**29.2 ± 0.0%**	**33.3 ± 8.3%**	**27.8 ± 4.8%**	**36.1 ± 4.8%**

* 100 attempts, miniF2F test set
† 50 attempts, 20% of miniF2F test set
‡ 50 attempts, 10% of miniF2F test set
§ 50 attempts, 5% of miniF2F test set

Table 1: Evaluation of LEGO-Prover and Draft-Sketch-Prove for a fixed number of prover attempts on various subsets of the miniF2F test set. Due to the cost, we reduce the number of attempts and evaluate more costly models on a subset of the test set. Consequently, LEGO-Prover and Draft-Sketch-Prove are comparable between rows, but not all columns can be directly compared. Standard deviations over 3 trials are reported.

4 A behavioural analysis of LEGO-Prover

Assuming there is an improvement, what aspect of LEGO-Prover is responsible? The most obvious explanation is that LEGO-Prover develops a library of useful lemmas. These may be broadly applicable and useful results that can be **directly reused** in the problem (as a mathematician might do when directly applying a theorem). Alternatively, these may be useful exemplars that are **softly reused** (i.e., fragments reproduced with minor edits). This behaviour is akin, e.g., to forking a code repository as the starting point for a new project.

This explanation hypothesizes two key behaviours: First, that LEGO-Prover can directly or indirectly leverage relevant lemmas in its proof writing; i.e., LEGO-Prover can **use** relevant lemmas. Second, that the library contains *general* lemmas that are both relevant to and leveraged in several tasks. I.e., LEGO-Prover can **reuse** lemmas.

In the following sections, we examine both direct and indirect reuse, finding evidence for use but against both soft and direct reuse.

4.1 Quantifying Soft Use

We quantify the soft use between a given input lemma and the PROVER's solution as a modified normalized Levenshtein distance (Levenshtein, 1966). For a lemma L and proof P then $\text{soft_use}(L, P) := 1 - \frac{|\text{LCS}(L,P)|}{|L|}$ where LCS is the longest common subsequence. Note the proof may contain an arbitrary number of additional tokens, as a used lemma is likely to be but a fragment of the final proof. See Appendix F for example lemmas, their soft use scores, and a more detailed description of soft use.

4.2 Does the PROVER use the input lemmas?

A prerequisite for tool reuse is tool use: a tool cannot be used several times if it is not used at all.

To test for direct use, we apply the methodology by Berlot-Attwell et al. (2024): for every retrieved lemma provided to the PROVER we directly check for the presence of the lemma (verbatim use) or the lemma's name (name use) in the PROVER's output (the latter being a generous way of accounting for changes such as whitespace or comments that do not impact semantics). We find that LEGO-Prover is capable of direct use with gpt-4o and gpt-4o-mini (i.e., directly copying an input lemma into the final proof – see Table 2). While we do not find direct use with o3-mini, a manual inspection of the instances of name use finds two solutions where the lemmas were copied from the input with some modifications (i.e., soft use).

Model	% Test Set	Verbatim Use	Name Use
GPT-4o	20%	$4.5 \pm 3.9\%$	$20.9 \pm 6.9\%$
GPT-4o	10%	$19.1 \pm 16.5\%$	$28.6 \pm 14.3\%$
o3-mini	10%	$0.0 \pm 0.0\%$	$23.6 \pm 12.0\%$
4o-mini	100%	$6.9 \pm 1.0\%$	$24.8 \pm 9.5\%$

Table 2: Percentage of found solutions in which a retrieved lemma or its name appears in the solution. While LEGO-Prover's PROVER can reproduce the retrieved input lemmas, Table 3 reveals these lemmas are not reused across tasks. We report average and standard deviation over 3 runs. Note that we run some models on random subsets of the miniF2F test set due to cost.

We also find evidence for indirect use. For any given threshold of the soft use score, we can calculate the proportion of input lemmas with a soft-use score of at least the threshold. This defines a survival function with the threshold as the independent variable. The greater the extent of soft use exhibited by the system, the slower we expect this curve to decay. For systems that perform direct use, this curve does not reach zero as the threshold approaches one. Instead, the lower bound is the proportion of lemmas that are directly used. To analyze soft-use behaviour, we compare the survival function of retrieved lemmas against that of other baseline populations of lemmas. The first base-

line is a pool of non-retrieved lemmas produced by LEGO-Prover (i.e., lemmas written by the same LLM for the same tasks but *not* retrieved and provided to the PROVER). These non-retrieved lemmas represent semantically and stylistically similar lemmas. The second baseline is a pool of unrelated human-written lemmas extracted from the 2019 Archive of Formal Proofs (AFP)[2].

These survival curves are plotted in the top row of Figure 1. As expected, we see that the survival curve of unrelated AFP lemmas drops fastest and rapidly reaches zero (demonstrating that our soft use metric is capturing relevant information). At high similarity thresholds, the retrieved lemmas maintain a larger population than the non-retrieved lemmas, thus providing evidence for soft use. Note that o3-mini is consistent with the trend, but has noisier measurements, as it was evaluated on a smaller dataset due to cost limitations.

4.3 Does LEGO-Prover learn a library?

The ability to use tools is a necessary but insufficient condition for library learning. Unlike mere task decomposition, library learning must create *reusable tools*. We must therefore inspect the reuse behaviour of LEGO-Prover. A lemma is said to be reused n times by the PROVER if it is used in $n+1$ proofs (i.e., the initial use, followed by n reuses).

Direct Reuse We find no evidence for direct reuse of lemmas (see Table 3). We find only one instance of name reuse in 189 successful proofs generated by gpt-4o-mini. Manual inspection reveals that this reuse does not appear in the final verified proof as the reuse was part of an erroneous proof step generated by the LLM that was replaced by the human-written proof correction heuristics.

Soft Reuse While these results bode poorly, the original authors claim that "[m]any skills cannot be directly reused but are very helpful as reference examples for formalizing the main problem." Therefore, to study LEGO-Prover's soft reuse behaviour, we repeat our survival curves from Section 4.2 but instead plot the proportion of lemmas used in at least two solutions generated by the LLM (see Figure 1, bottom row). In sharp contrast to our single-use findings there is no significant difference between the retrieved and non-retrieved lemmas. This suggests that LEGO-Prover learns overly-specific lemmas that are not reused in other problems.

As an alternative analysis, we also plot the survival function for the proportion of solved tasks exhibiting soft use, or soft reuse, at a given threshold (see Figure 2). We note that the soft reuse curves are within one standard deviation of 0% of tasks by the 70% soft-reuse threshold, and reach zero while 30-40% of tasks continue exhibiting soft use. This suggests that soft reuse is not occurring. Note that these curves also reveal an upper bound on the proportion of tasks that can benefit from lemma reuse, given the created library. Unlike our previous analyses, this bound arises from the *retriever* that selects lemmas from the library. Specifically, some tasks do not retrieve any lemmas that are also retrieved by another task. Therefore, it is impossible for these tasks to exhibit PROVER reuse. Consequently, at a soft use threshold of zero (i.e., all retrieved lemmas are considered to be soft used) the task survival functions for reuse still attains a value of less than 100%.

5 What drives LEGO-Prover's superior miniF2F performance?

Table 1 *appears* to demonstrate that LEGO-Prover outperforms Draft-Sketch-Prove. Wang et al. (2024a), naturally, claim that library learning is the mechanism for this improvement. *However*, we have shown that LEGO-Prover does not reuse the learned library. As such, we claim that the improvements instead arise from increased compute. The original data compares Draft-Sketch-Prove and LEGO-Prover based on a fixed number of PROVER attempts. This fails to account for the compute used by the EVOLVER, which runs in parallel to the PROVER. This compute is significant, as the default hyperparameters create twice as many EVOLVER processes than PROVER processes. Furthermore, the PROVER is more computationally intensive, as it performs a 2-stage process of decomposition and formalization. Using cost as a proxy for GPU compute, we estimate that each LEGO-Prover attempt uses approximately 6 times more GPU compute than Draft-Sketch-Prove, increasing to 14 times with o3-mini (see Table 4). Adjusting for the GPU budget, we find that Draft-Sketch-Prove matches or outperforms LEGO-Prover (see Table 5).

However, this analysis is limited – it is possible that Draft-Sketch-Prove plateaus before LEGO-Prover, or that LEGO-Prover requires enough itera-

[2] https://www.isa-afp.org/; see Appendix B for details.

Model	% of Test Set	Successful Attempts	Lemmas in Prompts	Verbatim reused		Name reused	
				1	2+	1	2+
GPT-4o	20%	43/5537	121	0	0	0	0
GPT-4o	10%	21/2727	64	0	0	0	0
o3-mini	10%	20/1825	75	0	0	0	0
4o-mini	100%	189/57419	583	0	0	1	0

Table 3: Lemma reuse in successful PROVER attempts. For various models and random subsets of the miniF2F test set, we report the proportion of successful PROVER attempts, the number of unique retrieved lemmas occurring in the PROVER's input prompts, the number of lemmas reused verbatim once, or more than once, and the number of lemmas whose *name* is reused once, or more than once. A lemma is reused N times if it appears in $N+1$ solutions (i.e., the initial use, and then N reuses). We sum all figures over 3 runs. This low reuse is also observed in the original logs released by Wang et al. (2024a); see Appendix, Table 11.

	gpt-4o-mini	gpt-4o	o3-mini
Cost Ratio	5.84	5.94	14.23

Table 4: We find that LEGO-Prover uses significantly more GPU resources that Draft-Sketch-Prove per attempt, and calculate the ratio of the costs under a fixed number of attempts. This ratio grows larger for o3-mini as LEGO-Prover uses significantly more output tokens per input token. We speculate that lemma creation and use requires a higher proportion of reasoning tokens than directly solving the task.

tions for the library to reach a critical mass. Therefore, we run a followup experiment in which we increase the number of Draft-Sketch-Prove iterations to approximately match the GPU resource usage, with cost as as proxy. Doing so, we find that Draft-Sketch-Prove remains within one standard deviation or outperforms LEGO-Prover throughout the evaluation (see Figure 3).

6 Cost-Corrected Behaviour in other ICL LLM Library Learning Systems

To test if our findings generalize we perform simple followup experiments on two other datasets and ICL library learning systems: AgentOptimizer (Zhang et al., 2024) on MATH (Hendrycks et al., 2021b) with gpt-4o-mini & gpt-4o, and TroVE (Wang et al., 2024d) on TabMWP (Lu et al., 2023) and MATH with CodeLlama-7b-Instruct. These datasets contain mathematical problems of differing complexity; TroVE and AgentOptimizer solve these tasks via Python code generation, each with a different approach. TroVE generates code conditioned on the Python signatures of the entire library; AgentOptimizer learns offline and exposes the library at inference time via ICL tool calling.

Adjusting TroVe's direct prompting majority-vote baseline to use a comparable computational budget, TabMWP and MATH performance is almost identical (see Table 6 for mean and standard deviation over 5 runs). See Appendix G for details.

We also reproduced AgentOptimizer's experimental methodology, again finding that AgentOptimizer uses equal or greater compute for comparable performance (see Table 7). In addition to the cost of offline library learning, the inference cost is also generally higher, presumably due to the tool specification (see Appendix, Tables 13 through 17 for cost details, and Appendix H for hyperparameters).

7 Related Work

Within the area of ICL LLM-based library learning, there has been growing evidence that these systems often do not perform direct reuse. Berlot-Attwell et al. (2024) previously studied LEGO-Prover and TroVE (Wang et al., 2024c), finding negligible direct reuse within these systems and proposing that the performance gains are due to self-correction and indirect reuse. Nguyen et al. (2024) reported that their library learning DynaSaur system achieved SoTA results on the GAIA agent benchmark (Mialon et al., 2024), but exhibited low direct reuse of the learned libraries. They did not speculate on the implications, beyond that it may be addressed via curriculum learning. Our work: (1) provides further evidence against direct reuse causing performance gains (to the best of our knowledge we are the first to evaluate direct reuse in a reasoning model), (2) advances our understanding by providing new evidence against the soft reuse hypothesis, and (3) provides a simple compute-based

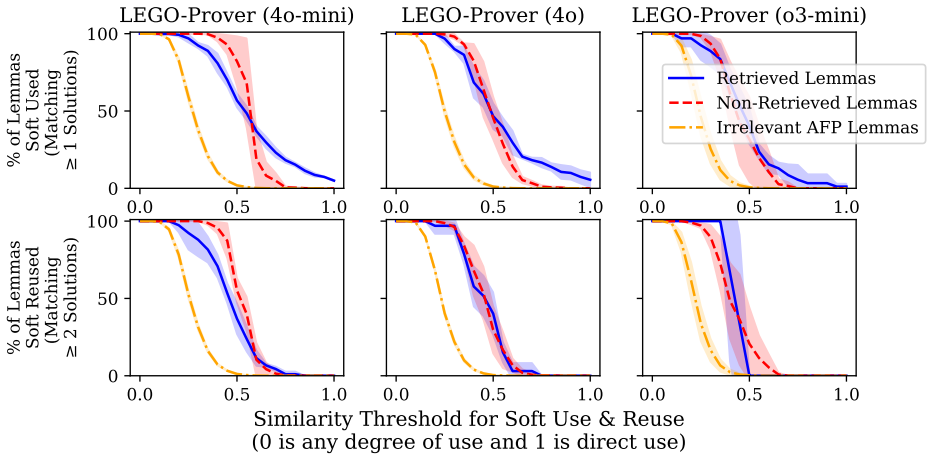

Figure 1: These plots demonstrate that LEGO-Prover exhibits soft use, but not reuse, of lemmas. The top row concerns use. It plots the percentage of lemmas meeting a given the threshold for soft use in at least one solution (see Section 4.1 for the definition of this score). The lemmas available to the PROVER are shown in the solid blue line, these are the only lemmas for which use is possible. As expected, in the top row this line remains the highest for large values of the threshold, surpassing the non-retrieved lemmas (red dashed line). However, in the bottom row we consider the proportion of lemmas meeting the soft use threshold in *two* tasks (i.e., reused by LEGO-Prover). Unlike in the top row, there is no significant difference between the lines – suggesting retrieved lemmas are only meaningful in a single task (consequently, used, not reused). See Section 4.2 for a detailed explanation.

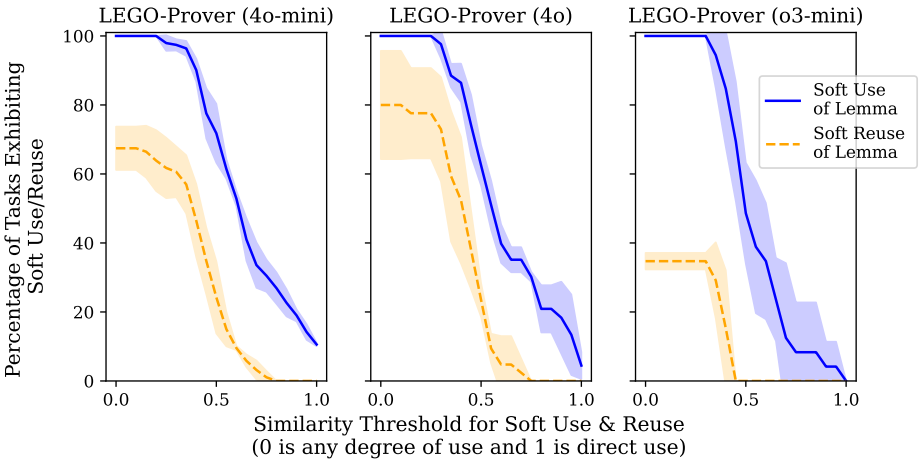

Figure 2: We define a normalized score for the degree to which a lemma is used in a proof (i.e., soft use). Using this score and a threshold, we can determine whether LEGO-Prover's solution to a task soft-uses the retrieved lemmas provided in the PROVER's context. In blue we plot the percentage of solutions meeting the desired threshold for soft use. In orange, we plot the percentage of solutions meeting the threshold in a lemma that is also used at the same level in at least one other solution. We observe that the lines for reuse rapidly reach zero for even moderate similarity thresholds wheras the lines for use remain relatively high. This suggests that lemmas are used by the PROVER in generating solutions, but that there tends to be only one task that exploits a given lemma.

hypothesis as to why these systems outperform their baselines – they increase test-time compute in a way unaccounted for in evaluation.

In the related domain of Python code generation, Olausson et al. (2024) studied the relative efficacy of self-correction as compared to increased sampling for the APPS (Hendrycks et al., 2021a) and HumanEval (Chen et al., 2021) tasks. Unlike our results, they found a moderate effect when accounting for compute. They hypothesized that "current models are held back by their inability to reliably produce accurate and useful feedback on why the code is wrong." Given the known success of tool-using LLMs (e.g., ViperGPT (Surís et al., 2023) achieved SotA performance on vision-language tasks using a code generating LLM con-

	gpt-4o-mini*	gpt-4o†	gpt-4o‡	gpt-4o§	o3-mini‡	o3-mini§
DSP	**35.9 ± 0.9%**	23.1 ± 1.2%	**25.0 ± 0.0%**	**25.0 ± 0.0%**	**19.4 ± 2.4%**	**19.4 ± 4.8%**
LP	17.5 ± 0.9%	**23.8 ± 1.2%**	22.2 ± 2.4%	22.2 ± 4.8%	15.3 ± 2.4%	**19.4 ± 9.6%**
LP attempts	17	9	8	6	4	3

* 100 DSP attempts, miniF2F test set
† 50 DSP attempts, 20% of miniF2F test set
‡ 50 DSP attempts, 10% of miniF2F test set
§ 50 DSP attempts, 5% of miniF2F test set

Table 5: A modification of Table 1 that limits the number of LEGO-Prover attempts to match Draft-Sketch-Prove's compute budget. Note that Draft-Sketch-Prove consistently outperforms LEGO-Prover or is within one standard deviation of LEGO-Prover's performance. Average accuracy and standard deviations are again reported over 3 trials.

Dataset	Compute Matched Baseline	TroVE Reported (best of 5)
Algebra	**0.27 ± 0.01**	0.25
Counting	0.24 ± 0.00	**0.26**
Geometry	**0.08 ± 0.01**	0.08
Intermediate	**0.14 ± 0.01**	0.11
Number	**0.28 ± 0.02**	0.25
Prealgebra	**0.33 ± 0.02**	0.29
Precalculus	0.15 ± 0.01	**0.17**
TabMWP	0.45 ± 0.01	**0.47**

Table 6: TroVE's MATH and TabMWP performance is comparable to the baseline given equivalent compute. We report mean and standard deviation over 5 runs.

MATH Geometry		
4o-mini	Agent Opt.	ReAct
Accuracy	60.00 ± 2.50%	60.00 ± 2.50%
Cost	\$0.77 ± 0.09	\$0.09 ± 0.02
4o	Agent Opt.	ReAct
Accuracy	64.17 ± 1.44%	66.25 ± 2.50%
Cost	\$11.22 ± 0.47	\$1.28 ± 0.19
MATH Precalculus		
4o-mini	Agent Opt.	ReAct
Accuracy	58.75 ± 2.50%	58.75 ± 3.75%
Cost	\$0.81 ± 0.06	\$0.15 ± 0.02
4o	Agent Opt.	ReAct
Accuracy	61.67 ± 4.02%	63.33 ± 4.02%
Cost	\$10.94 ± 0.47	\$1.87 ± 0.20

Table 7: AgentOptimizer compute costs significantly exceed the ReAct baseline for comparable performance. We report mean and standard deviation over 3 runs.

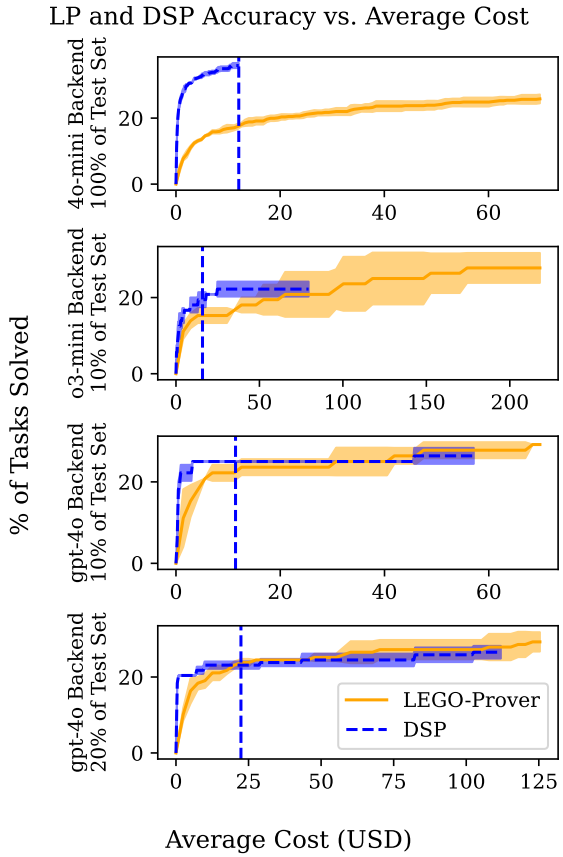

Figure 3: Average LEGO-Prover and Draft-Sketch-Prove performance compared by average cost in USD. The shaded region is 1 standard deviation (3 trials). Draft-Sketch-Prove has comparable or superior performance. The vertical line is the number of Draft-Sketch-Prove prover attempts equal to LEGO-Prover's. Due to cost, o3-mini was run on a random 10% test subset; gpt-4o was run on 10% and 20% subsets. As o3-mini cost over twice our predicted cost, we were unable to run Draft-Sketch-Prove for a comparable budget.

ditioned on a human provided Python library for vision processing), we suspect that there may be a similar bottleneck in library learning. I.e., these systems may be held back by an inability to reliably produce general theorems and would improve in performance if given human-created libraries.

In the domain of agentic AI, Kapoor et al. (2024) call for the use of cost as a metric and claim that planning, reflection, and debugging offer no improvement over trivial baselines on the HumanEval (Chen et al., 2021) task. Our work provides similar findings in the area of autoformalization when using the techniques of library learning. This is interesting, as autoformalization is closer to a translation task rather than generation (i.e., converting a natural language proof to code) and, unlike Python programming, we have access to a strong verifier that can confirm the correctness of our reasoning (i.e., unlike self-correction that may introduce errors by fixing hallucinatory errors, all intermediary lemmas are verified by Isabelle). Despite these advantages, we do not to find an improvement in accuracy. Furthermore, given our focus on library learning specifically, we also go into the details of whether the system internally behaves as expected, as opposed to studying the general trends of cost and accuracy. In doing so, we find evidence that these techniques are ineffective and fail to operate as believed, suggesting that there are fundamental flaws in these specific approaches.

8 Conclusions

We have gained a number of insights into the LEGO-Prover system. Firstly, LEGO-Prover does not exhibit reuse behaviour – therefore, it does not perform library learning. Secondly, LEGO-Prover does not improve performance over a prompting baseline once computational cost is accounted for.

These findings, particularly with respect to accounting for computational budget, become wide-reaching when considering our followup experiments with TroVE and AgentOptimizer, as well as the broader ICL LLM library learning literature. Our work combines with other emerging evidence to suggest serious flaws in the assessment of ICL LLM-based library learning. It was believed that LLM library learning works, as it improves performance – but our findings on the importance of accounting for computational cost cast this into question. We are unaware of *any* LLM-based library learning work that controls for compute in its evaluation. Consequently, the observed benefits of library learning may be due to hidden inference-time scaling. This calls into the question the efficacy of these works – and we cannot hope to make progress without resolving this problem. This would also go towards explaining other findings: previous work has demonstrated that the library learning system TroVE does not outperform library-free ablations on MATH (Berlot-Attwell et al., 2024) – our work both explains this finding (i.e., Berlot-Attwell et al. (2024)'s ablation implicitly matched computational cost), and further demonstrates that direct prompting alone has comparable TabMWP performance. Our work also explains apparent paradoxes – the DynaSaur library learning system reports SoTA performance *and* a lack of reuse behaviour (Nguyen et al., 2024). This contradiction is resolved if increased inference-time compute is responsible for the performance gains. Indeed, in their evaluation, they fixed all models to 20 steps. I.e., their evaluation did not account for the differences in computational cost per step between DynaSaur, direct prompting of gpt-4o, Sibyl (Wang et al., 2024b), and HF Agent (Roucher and Nguyen, 2024).

These findings also call into question the argument that LLM library learners can reduce costs (at least in their current form). The caching of generated tools is moot if said tools are not reused, and our analysis suggests that they are not. Secondly, the savings from caching must exceed the cost of creating and selecting these tools – we perform the first cost-controlled comparison of an LLM library learner that we are aware of and find no improvement over prompting with the same budget.

Looking to how we can build better systems – from the perspective of accuracy, our work suggests that our effort may be best directed towards improving base models. From the perspective of library learning, our work calls into question the innate ability of LLMs to perform refactoring using in-context learning alone. Therefore, we suggest incorporating symbolic refactoring systems that are known to work (e.g., the LILO (Grand et al., 2024) system takes this approach). If we instead wish to use LLMs directly for library learning then, as context learning is not enough, it may be worthwhile to explore finetuning or RL (e.g., via GRPO (DeepSeek-AI et al., 2025) with human-written reward functions, possibly based on compression). Critically, any such system must be evaluated thoroughly. Therefore, any system claiming to perform library learning should be evaluated with respect to both accuracy **accounting for computational budget**, and the system's behaviour. The extent of reuse must be measured, be it direct or soft reuse. In service of this goal, we also suggest the construction of a synthetic diagnostic dataset based on a known ground truth library.

9 Limitations

For reasons of cost, o3-mini and gpt-4o were run on random subsets of miniF2F. Despite this, we observe the same LEGO-Prover trends across all three LLMs studied, and gpt-4o-mini was evaluated on the full test set. We release all our collected data and code to facilitate analysis and adaptation to other LLMs. We attempted evaluation with the open source `DeepSeek-R1-Distill` family of models, but found that all posses a strong bias for Lean 4 code. When the models did produce Isabelle code, it was still produced within a `lean` code block; based on this we suspect that lean theorem proving was used in the distillation data. Our TroVE experiments are performed with an open source LLM (Code-Llama-7b-Instruct).

Figure 3 is plotted using the average cost per attempt (see Appendix E for details). This is a slight distortion as, under Draft-Sketch-Prove, the true cost per attempt reduces over time, as the pool of unsolved problems tackled per attempt shrinks.

LEGO-Prover uses `text-embedding-ada-002` for retrieval. For simplicity we omit the cost of Ada from our plots – said cost was quite low and as the omission reduces LEGO-Prover's running costs this does not impact our findings.

We performed our experiments using the released LEGO-Prover code base; this code base initializes the EVOLVER with a database of all miniF2F test queries even when evaluating a subset of miniF2F. We tested the impact by evaluating LEGO-Prover on a small subset using gpt-4o-mini and found that preventing the EVOLVER from retrieving these entries *slowed* the rate at which problems were solved. Regardless, given the vast gap in cost (persisting in our gpt-4o-mini experiments where the EVOLVER was initialized correctly), we do not believe this impacts on our results.

The key assumption in our soft use score is that soft use takes a form similar to the modification of starter code. Other forms of soft reuse divorced from syntax (e.g., involving heavy paraphrasing) would not be detected. Having said this, as the difference between the lemma and the proof increases the likelihood of the introducing an error increases (as the model further distances itself from the Isabelle-verified code); thus the value of such reuse would decrease.

Another limitation is that because our metric does not penalize the insertion of new tokens, it likely scores short lemmas higher on average. We believe this trade-off is acceptable given the premise that the final proof would (and in any reasonable library learning system *should*) contain more than just the lemma. E.g., a proof may use the Pythagorean theorem, but in conjunction with other axioms, lemmas, and reasoning steps. This also allows correctly prevents penalizing the addition of comments or the interlacing of proof steps.

One concern is that gpt-4o and o3-mini scores appear low on our random test subsets. We suspect the random test sets consist of harder problems. We can observe in Tables 1 & 10 that while Draft-Sketch-Prove reproduces Wang et al. (2024a)'s reported overall miniF2F test set accuracy, its performance drops by 15.9% (absolute) on our random 5% subset of the test set. We are also confident in our overall findings as low reuse is also observed in the original experiment logs released by Wang et al. (2024a) (see Appendix, Table 11).

In our initial experiments, o3-mini struggled to generate solutions inside of markdown code blocks. Prompt tuning significantly improved the reliability, but it remains imperfect – this behaviour may reduce LEGO-Prover performance by slowing the EVOLVER. Addressing prompts more generally, neither Draft-Sketch-Prove nor LEGO-Prover were designed for reasoning models; consequently their prompts are suboptimal as OpenAI recommends short prompts without examples. To ensure fair cross-comparison between models and proof formalization systems, we chose to make only the minimal prompt changes required to correct any gross errors of behaviour in Draft-Sketch-Prove or LEGO-Prover (e.g., formatting errors, producing proofs for example problems, etc...). Based on our prompt stability experiments (see Appendix C) we believe that minor rephrasing of our final prompts would not change our results.

References

Ian Berlot-Attwell, Frank Rudzicz, and Xujie Si. 2024. Library learning doesn't: The curious case of the single-use "library". In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*.

Anwoy Chatterjee, H S V N S Kowndinya Renduchintala, Sumit Bhatia, and Tanmoy Chakraborty. 2024. POSIX: A prompt sensitivity index for large language

models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 14550–14565, Miami, Florida, USA. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *CoRR*, abs/2501.12948.

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. 2021. DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 835–850. ACM.

Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. 2024. LILO: Learning interpretable libraries by compressing and documenting code. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021a. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

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

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. AI agents that matter. *CoRR*, abs/2407.01502.

Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A. Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, Franck Dernoncourt, and Tianyi Zhou. 2024. DynaSaur: Large language agents beyond predefined actions. *CoRR*, abs/2411.01747.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations (ICLR)*.

OpenAI. 2024. GPT-4o mini: advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2025-03-19.

Lawrence C. Paulson. 1994. *Isabelle - A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer.

Cheng Qian, Chi Han, Yi Ren Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 6922–6939. Association for Computational Linguistics.

Aymeric Roucher and Dang Nguyen. 2024. Beating GAIA with Transformers Agents. https://github.com/aymeric-roucher/GAIA. Accessed: 2025-03-19.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual inference via python execution for reasoning. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 11854–11864. IEEE.

John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.

Jan Philip Wahle, Terry Ruas, Yang Xu, and Bela Gipp. 2024. Paraphrase types elicit prompt engineering capabilities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11004–11033, Miami, Florida, USA. Association for Computational Linguistics.

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. 2024a. LEGO-Prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. 2024b. Sibyl: Simple yet effective agent framework for complex real-world reasoning. *CoRR*, abs/2407.10718.

Zhiruo Wang, Graham Neubig, and Daniel Fried. 2024c. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Zhiruo Wang, Graham Neubig, and Daniel Fried. 2024d. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Forty-first International Conference on Machine Learning*.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. 2024. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Offline training of language model agents with functions as learnable weights. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 60315–60335. PMLR.

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

A LEGO-Prover Hyperparameters

In our experiments we use gpt-4o version 2024-08-06, o3-mini version 2025-01-31, gpt-4o-mini version 2024-07-18, and text-embedding-ada-002.

LEGO-Prover's hyperparameters are listed in Table 8; values changed from defaults are underlined. Note that Wang et al. (2024a) use 50 attempts for their ablation experiments. Draft-Sketch-Prove's hyperparameters are listed in Table 9. Note that we modified Draft-Sketch-Prove to run 11 processes in parallel (with each launching its own Isabelle instance) so that the average resources per Isabelle instance is equal to LEGO-Prover. The prompts used in our experiment are documented in Appendix D. All our experiments use Isabelle 2022.

Each experiment is performed with a resource allocation of 180 GB of RAM and 50 Intel Broadwell CPU cores @2.095 GHz. These resources are shared among all Isabelle instances. Approximately 4.5 CPU-days was used over the course of all experiments.

B Data Details

We used fixed, random, non-overlapping subsets of the miniF2F test set for our experiments (5%, 10% and 20%). For reasons of cost, we only study system behaviour with human-written English proofs as inputs to the systems.

Our pool of unrelated Isabelle lemmas used in Figures 1 and 2 are extracted from the `2019-08-19` Archive of Formal Proofs[3]. Specifically, we used all matches to the following Python3 regex:

```
(?:lemma|theorem)\s+[^:]+?\s*:(?:[\s\S]
(?!lemma|theorem))+?qed
```

This regex matches lemmas and theorems with similar syntax to the in-context examples, though it may prematurely truncate some lemmas and theorems.

C Stability with Respect to Prompt

For both Draft-Sketch-Prove and LEGO-Prover we were required to make minor modifications to the prompt to ensure that gpt-4o-mini generated responses with the correct formatting (See Appendix D for the prompts and our modifications). To address the concern that our choice of prompt may be disadvantaging LEGO-Prover, we generate five prompt paraphrases, and following Wahle et al. (2024) study the relative stability under the paraphrased prompts versus the original prompt. We generate the prompt paraphrases using GPT3.5, as done by Chatterjee et al. (2024).

We find no evidence that LEGO-Prover is disadvantaged by prompt: accuracies are within one standard deviation, and there was no maximum potential gain (i.e., no problems were solved under one of the paraphrases, but not by the original

[3]Available for download at the time of publication from https://sourceforge.net/projects/afp/files/afp-Isabelle2019/

LEGO-Prover Hyperparameter	Value
Temperature	0.7, (N/A for o3-mini)
# PROVER Processes	3
# EVOLVER Processes	8
# Attempts	100 for full miniF2F test set, else 50
EVOLVER maximum tokens decoded	1024, (8092 for o3-mini)
PROVER maximum tokens decoded	2000, (8092 for o3-mini)

Table 8: LEGO-Prover hyperparameters. Values changed from Wang et al. (2024a) are underlined.

Draft-Sketch-Prove Hyperparameter	Value
Temperature	0.0, (N/A for o3-mini)
# Processes	11
# Attempts	100 for full miniF2F test set, else 50
Maximum tokens decoded	2048, (8092 for o3-mini)

Table 9: Draft-Sketch-Prove hyperparameters. Values changed from Jiang et al. (2023) are underlined.

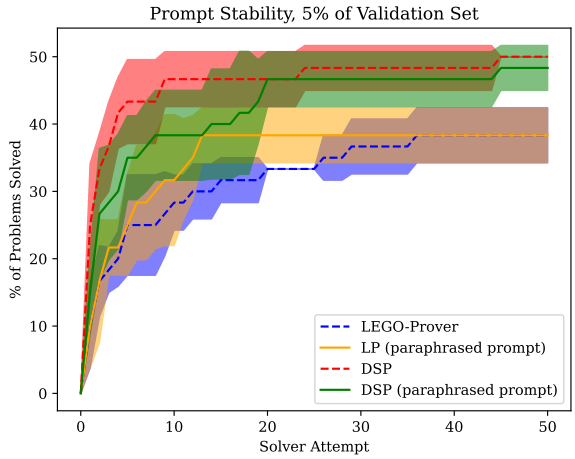

Figure 4: For LEGO-Prover and Draft-Sketch-Prove we compare 5 runs with our default prompt versus 5 runs with different paraphrased prompts. We evaluate on a subset of the validation set. We find that the systems are fairly robust to minor changes in wording – performance is typically within 1 standard deviation. These results are for a 5% subset of the validation split.

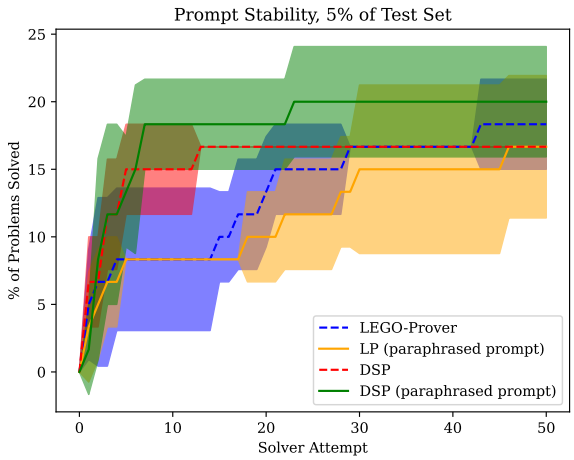

Figure 5: For LEGO-Prover and Draft-Sketch-Prove we compare 5 runs with our default prompt versus 5 runs with different paraphrased prompts. We evaluate on a subset of the test set. We find that these system are fairly robust to minor changes in wording – performance is typically within 1 standard deviation. These results are for a 5% subset of the test split.

prompt). See Table 10 for details. Indeed, Draft-Sketch-Prove exhibited potential gain, suggesting that our Draft-Sketch-Prove prompt may be supoptimal. We also plot the performance per attempt in Figures 4 and 5.

Note that we modify (Wahle et al., 2024)'s maximum potential gain metric to reflect that we collect data from 5 baseline trials instead of one. In our implementation, we consider it to be a 0% improvement for a specific problem if a paraphrase achieves

model	val Baseline acc	val Paraphrase acc	Max Pot. Gain	Max Pot. Loss
dsp	$50.0 \pm 0.0\%$	$48.3 \pm 3.7\%$	0.0%	0.0%
lp	$38.3 \pm 4.6\%$	$38.3 \pm 4.6\%$	0.0%	0.0%
model	**test Baseline acc**	**test Paraphrase acc**	**Max Pot. Gain**	**Max Pot. Loss**
dsp	$16.7 \pm 0.0\%$	$20.0 \pm 4.6\%$	16.7%	0.0%
lp	$18.3 \pm 3.7\%$	$16.7 \pm 5.9\%$	0.0%	0.0%

Table 10: Average test accuracy of the models under both our baseline prompt and paraphrases of the prompt. We also report variability in the results, following the methodology for prompt sensitivity from Wahle et al. (2024). All experiments were run using 50 attempts, 5 runs, and on a 5% split of the miniF2F validation and test sets respectively.

the same performance or worse (i.e., no improvement is possible for problems solvable under the baseline prompt), and 100% if it solves a task that was unsolved under the baseline prompt. Averaging over all problems this becomes the fraction of problems that are only solved by a paraphrased prompt. The maximum potential loss is calculated in the same way but with the rewards reversed, thus becoming the fraction of problems that are solved by the baseline prompt, but by none of the paraphrased prompts.

D Changes to Prompts

D.1 Draft-Sketch-Prove

Draft-Sketch-Prove was originally designed for text completion models, specifically Codex `code-davinci-002` (Hendrycks et al., 2021a). As it was not designed for conversational models, and none of the models evaluated support a text completion mode, we developed our own minimal system prompts to preface the few-shot examples preprended by Draft-Sketch-Prove to the task statement.

D.1.1 gpt-4o-mini & gpt-4o System Prompt

In the prompt (see Figure 6) we emphasized the use of sledgehammer as it was a focus of the original Draft-Sketch-Prove paper, and the model would avoid its use unless told otherwise. After designing the prompt for gpt-4o-mini, we found that gpt-4o displayed no obviously incorrect behaviours, so we did not modify it further. We validated our prompts on a subset of two validation set problems (`mathd_algebra_109` and `algebra_sqineq_2at2pclta2c2p41pc`).

D.1.2 o3-mini System Prompt

For o3-mini we modified the prompt (see Figure 7) to explicitly prohibit o3-mini from answering the in-context examples. We also removed the instruction to use sledgehammer as o3-mini would ignore the examples and attempt to use "`by sledgehammer`" instead of "`sledgehammer`" – this would cause parsing errors in the human proof-correcting heuristics. Based on our small validation set of two problems, we found that removing the instruction to use sledgehammer worked more reliably than adding instructions to use sledgehammer correctly.

D.2 LEGO-Prover

LEGO-Prover was designed for conversational models, and as such has default system prompts for the PROVER decomposer, PROVER formalizer, and several EVOLVER variants. As a general rule, we attempted to minimize changes to these prompts where possible, doing so only when we found erroneous behaviour. Text in green is added to the prompt, text in red is removed from the original prompt.

D.2.1 gpt-4o-mini prompts

For gpt-4o-mini we found it necessary to add additional formatting instructions to the PROVER's decomposer prompt (see Figure 8).

We also observed that the rate of rejection of proof attempts was high due to slight paraphrasing of the requested theorem statement. We addressed this by modifying the PROVER's formalizer prompt (see Figure 9).

The EVOLVER prompts were unchanged.

D.2.2 gpt-4o prompts

For gpt-4o we used the gpt-4o-mini prompts and further modified the PROVER's decomposer prompt to prevent it from decomposing the provided in-context examples (see Figure 10).

D.2.3 o3-mini prompts

For o3-mini we found it necessary to change all prompts to prevent the model from solving the in-

Model	% of Test Set	Successful Attempts	Lemmas in Prompts	Verbatim reused		Name reused	
				1	2+	1	2+
GPT-4o	20%	43/5537	121	0	0	0	0
GPT-4o	10%	21/2727	64	0	0	0	0
o3-mini	10%	20/1825	75	0	0	0	0
4o-mini	100%	189/57419	583	0	0	1	0
LP (human-test)	100%	122/?	339	1	0	2	0
LP (human-valid)	N/A	135/?	265	0	0	1	0
LP (gpt-test)	N/A	111/?	255	0	0	2	0
LP (gpt-valid)	N/A	127/?	374	0	0	1	0

Table 11: Extension of Table 3, reuse in successful PROVER attempts. The bottom half is extended with numbers calculated from the original LEGO-Prover logs released by Wang et al. (2024a).

As a mathematician familiar with Isabelle, your task is to complete the proof sketch provided by the user, MAKING SURE TO **ALWAYS** USE ‘sledgehammer‘ TO PROVE THE INTERMEDIATE RESULTS.

Figure 6: Draft-Sketch-Prove System Prompt for gpt-4o-mini & gpt-4o.

Complete the Isabelle proof sketch provided by the user. The user will provide several examples, but you should only complete the last example (i.e., the example that does not have a proof).

Figure 7: Draft-Sketch-Prove SystemPrompt for o3-mini.

As an mathematician and expert in isabelle theorem prover, your task is to analyze the given theorem (including problem’s informal statement, human written informal proof, and formal statement). Provide a better structured step by step proof that closer to isabelle. and request relevant lemmas, theorems that might help in proving this problem.
You will format your answer as follows:
- A section with the header ‘## Structured informal proof‘ containing the step-by-step proof.
- A subsequent section with the header ‘## Request skills‘ containing any requested relevant lemmas or theorems
- Each requested skill is represented by an isabelle code block containing its formal definition

Figure 8: LEGO-Prover: PROVER decomposer prompt for gpt-4o-mini.

As a mathematician familiar with Isabelle, your task is to provide a formal proof in response to a given problem statement.
Your proof should be structured and clearly written, meeting the following criteria:
- It can be verified by Isabelle.
- Each step of the proof should be explained in detail using comments enclosed in "(*" and "*)".
- The explanation for each step should be clear and concise, avoiding any unnecessary or apologetic language.
- You are **strongly encouraged** to create useful and reusable lemmas to solve the problem.
- The lemmas should be as general as possible (generalizable), and be able to cover a large step in proofs (non-trivial).
- You **MUST** copy the input lemma EXACTLY: **without changing whitespace**, without paraphrasing, and without adding comments.
Please ensure that your proof is well-organized and easy to follow, with each step building upon the previous one.

Figure 9: LEGO-Prover: PROVER formalizer prompt for gpt-4o-mini.

As an mathematician and expert in isabelle theorem prover, your task is to analyze the given theorem (including problem's informal statement, human written informal proof, and formal statement). Provide a better structured step by step proof that is closer to isabelle. You will be given several examples; you must only decompose the last theorem given (i.e., the only theorem without a provided solution).
Furthermore, you must and request relevant lemmas, theorems that might help in proving this problem.
You will format your answer as follows:
- A section with the header '## Structured informal proof' containing the step-by-step proof.
- A subsequent section with the header '## Request skills' containing any requested relevant lemmas or theorems
- Each requested skill is represented by an isabelle code block containing its formal definition

Figure 10: LEGO-Prover: PROVER decomposer prompt for gpt-4o.

context examples. We also found o3-mini struggled with placing the solution to proofs inside of Markdown code blocks, even after Markdown formatting was enabled by prepending "Formatting re-enabled" to the prompts.

For PROVER's decomposer prompt we re-enabled Markdown formatting and added instructions to not decompose the in-context examples (see Figure 11).

The PROVER's formalizer prompt was modified to re-enable Markdown formatting and to emphasize the desired output format (see Figure 12).

The evolver prompts were likewise modified (see Figure 13 through Figure 17). The do_request prompt (see Figure 13) was particularly problematic as o3-mini tended to disregard the formatting instructions.

E Calculation of LEGO-Prover Cost

In our experiments we recorded the total token usage per run. I.e., if an experiment run consisted of 100 attempts, then we recorded the cost of 100 attempts. However, to create Table 5 we required the cost for a reduced number of attempts. To estimate this, we determined the average cost per problem attempt. I.e., we calculated the total cost over all runs and divided this by the total number of problem attempts over all runs. For example, the 3 runs of the gpt-4o-mini experiment on the full test set used 100 attempts per problem each; therefore, we divided the total cost of these 3 experiment runs by 300 to determine the average cost per attempt.

Similarly, Figure 3 required the total expenditures up to the nth success, and we again estimated this using the average cost per attempt. Specifically, we plotted (x, y) points where x was the cost of i attempts on all remaining problems, and y was the number of problems solved after i attempts.

For further details, the code used to calculate Table 5 and plot Figure 3 is publicly available in our GitHub repository.

F Soft Use Score Examples

We quantify the soft use between a given input lemma and the PROVER's solution as the minimum proportion of space-separated tokens in the lemma that must be deleted or substituted for the lemma to appear within the solution.

We obtain this score by first computing the Levenshtein distance (Levenshtein, 1966) (i.e., the minimum number of deletions, replacements, or insertions required to convert the lemma into the proof) assigning a weight of 0 to insertions. We allow for an arbitrary number of insertions, as a used lemma is likely to be only a fragment of the final proof. This modified Levenshtein distance thus has a value between 0 (all tokens in the lemma also appear in the solution and in the same order) and, N, the number of tokens in the lemma (no tokens in the lemma appear in the solution). We divide this value by the length of the lemma to produce a normalized score for soft use. Equivalently, for a lemma L and proof P then soft_use$(L, P) = 1 - \frac{|\text{LCS}(L,P)|}{|L|}$ where LCS is the longest common subsequence.

In this section we report examples of various retrieved lemmas and their soft use scores. The example chosen (`data/full_data/test/amc12a_2021_p9.json`) was generated in the first run of gpt-4o-mini; it was selected as it has the largest range of soft use values.

The English description for this task is:

> Which of the following is equivalent to $(2+3)(2^2+3^2)(2^4+3^4)(2^8+3^8)(2^{16}+3^{16})(2^{32}+3^{32})(2^{64}+3^{64})$?
> **(A)** $3^{127}+2^{127}$ **(B)** $3^{127}+2^{127}+2\cdot 3^{63}+3\cdot 2^{63}$ **(C)** $3^{128}-2^{128}$ **(D)** $3^{128}+2^{128}$
> Show that it is **(C)** $3^{128}-2^{128}$.

The model output prior to correction by heuristics is reproduced in Figure 18. Some lines have been word wrapped to fit within margins.

The retrieved lemma in Figure 19 provided to the PROVER when generating the solution in Figure 18 attained a soft use score of 0.378. It is largely irrelevant, as it is specialized to a function that does not appear in this problem.

The retrieved lemmas in Figures 20 and 21 provided to the PROVER when generating the solution in Figure 18 attained soft use scores of 0.695 and 0.706 respectively. They are more relevant, involving powers of powers, but are again overly specialized and the proof tactics used (`simp add: power_add`, `simp add: power2_eq_square`, etc...) are not used in the PROVER's outputted solution. Note that LEGO-Prover's EVOLVER often produces minor variations of the same lemma as seen in this case.

The retrieved lemma in Figure 22 provided to the PROVER when generating the solution above attained a soft use score of 0.96. Note that some lines are word wrapped to fit within the margins.

Formatting re-enabled
As an mathematician and expert in isabelle theorem prover, your task is to analyze the given theorem (including problem's informal statement, human written informal proof, and formal statement). Provide a better structured step by step proof that closer to isabelle. and request relevant lemmas, theorems that might help in proving this problem. Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a structured step by step proof and requests for relevant lemmas).
You will format your answer as follows:
- A section with the header '## Structured informal proof' containing the step-by-step proof.
- A subsequent section with the header '## Request skills' containing any requested relevant lemmas or theorems
- Each requested skill is represented by an isabelle code block containing its formal definition

Figure 11: LEGO-Prover: PROVER decomposer prompt for o3-mini.

Formatting re-enabled
As a mathematician familiar with Isabelle, your task is to provide a formal proof in response to a given problem statement.
Your proof should be structured and clearly written, meeting the following criteria:
- It can be verified by Isabelle.
- Each step of the proof should be explained in detail using comments enclosed in "(*" and "*)".
- The explanation for each step should be clear and concise, avoiding any unnecessary or apologetic language.
- You are **strongly encouraged** to create useful and reusable lemmas to solve the problem.
- The lemmas should be as general as possible (generalizable), and be able to cover a large step in proofs (non-trivial).
- Your solution should be written within a Markdown isabelle code block. i.e., within a triple back ticked code block, labelled as "isabelle"
- You **MUST** copy the input lemma EXACTLY: **without changing whitespace**, without paraphrasing, and without adding comments. - Within the code block, you must create a new Isabelle 'theory' containing the aforementioned input lemma copied from the input, as well as your proof and any lemmas you created for the proof. Please ensure that your proof is well-organized and easy to follow, with each step building upon the previous one.

Figure 12: LEGO-Prover: PROVER formalizer prompt for o3-mini.

Formatting re-enabled
As a mathematician familiar with Isabelle, your task is to provide a formal proof in response to a given formal statement.
Your proof should be structured and clearly written, meeting the following criteria:
- It can be verified by Isabelle.
- Please ensure that your proof is well-organized and easy to follow, with each step building upon the previous one.
- THIS IS **CRITICAL**: YOUR PROOF **MUST** BE WRITTEN WITHIN A MARKDOWN ISABELLE CODE BLOCK!!! i.e., within a triple back ticked code block, labelled as "isabelle"

Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a formal proof).

{examples}

####################

Statement:
```isabelle
{formal_statement}
```

Proof

Figure 13: LEGO-Prover: EVOLVER do_request prompt for o3-mini.

It is reproduced almost exactly in the PROVER's output, with minor changes to parenthesization and the name of the lemma.

G Compute-Matched TroVE Evaluation

TroVE (Wang et al., 2024d) is a library learning method for solving a stream of tasks by generating Python code that exploits a library of functions. Said library is learned in an online fashion. TroVE works by generating 15 candidate solutions and performing majority vote, with functions from successful candidates being added to the library. More specifically, these 15 solutions are produced via 3 different prompts: 5 are generated via the SKIP prompt (directly prompting the LLM to generate a program), the CREATE prompt (prompt the LLM to generate a helper function to add to the library as well as a program that uses the helper to solve the problem), and the IMPORT prompt (generate a solution conditioned on the entire library). Functions created by accepted CREATE solutions are added to the library, which is periodically trimmed based on function usage statistics.

We perform compute-matching via a trivial mechanism: we compare baseline TroVE (5 samples from 3 prompts) with an all SKIP ablation (15 samples from the SKIP prompt). Before running our final experiments we tested that this approximately matched compute by counting token usage for TroVE and the ablation on a subset of TabMPW. Specifically, we found that TroVE used an average of 17457.61 tokens per problem, with the ablation using 15194.51 tokens per problem (only 87.1% as many tokens). We chose not to increase K further as we found that while the ablation only used 77% as many input tokens, it did consume slightly more output tokens (106%). The increase in output tokens is likely because TroVE's SKIP prompt invokes the LLM in text completion mode, often causing it to generate new, hallucinated TabMWP problems after generating the solution to the problem.

As per the original paper, we evaluate on TabMWP using `CodeLlama-7B-Instruct`.

Taking the maximum over 5 runs, our ablation achieves 0.46 compared to the TroVE performance of 0.47 reported by Wang et al. (2024d) (note that the choice of maximum rather than average

Formatting re-enabled
As an expert mathematician who is proficient in Isabelle theorem proving, your task is to modify the given lemma, theorem, function or definition given in the code to aid in solving one or more of the problems provided. Your should accomplish this by Extend Dimensions: If the problem is defined in a specific number of dimensions, consider if it would still hold in more or fewer dimensions.
Your answer **MUST** be written within a Markdown isabelle code block. i.e., within a triple back ticked code block, labelled as "isabelle"

Here is some reference problems, you should evolve the skill to help solving theses problems:
{problems}

Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a modified version).

############ Extend Dimensions ############
{examples}

####################
Skill to evolve
```isabelle
{code}
```

Evolved skill

Figure 14: LEGO-Prover: EVOLVER "extend dimensions" prompt for o3-mini.

Formatting re-enabled
As an expert mathematician who is proficient in Isabelle theorem proving, your task is to modify the given lemma, theorem, function or definition given in the code to aid in solving one or more of the problems provided. Your should accomplish this by ‘Identifying Key Concepts‘: Extract the essential ideas, methods, or theorems that are critical to solving the problem.
Your answer **MUST** be written within a Markdown isabelle code block. i.e., within a triple back ticked code block, labelled as "isabelle"

Here is some reference problems, you should evolve the skill to help solving theses problems:
{problems}

Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a modified version).

############ Identifying key Concepts ############
{examples}

####################
Skill to evolve
“‘isabelle
{code}
“‘

Evolved skill

Figure 15: LEGO-Prover: EVOLVER “identify key concepts” prompt for o3-mini.

Formatting re-enabled
As an expert mathematician who is proficient in Isabelle theorem proving, your task is to modify the given lemma, theorem, function or definition given in the code to aid in solving one or more of the problems provided. Your should accomplish this by Parameterize: If the problem involves specific numbers, generalize it by replacing these with variables.
Your answer **MUST** be written within a Markdown isabelle code block. i.e., within a triple back ticked code block, labelled as "isabelle"

Here is some reference problems, you should evolve the skill to help solving theses problems:
{problems}

Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a modified version).

############ Parameterize ############
{examples}

####################
Skill to evolve
```isabelle
{code}
```

Evolved skill

Figure 16: LEGO-Prover: EVOLVER "parameterize" prompt for o3-mini.

Formatting re-enabled
As an expert mathematician who is proficient in Isabelle theorem proving, your task is to modify the given lemma, theorem, function or definition given in the code to aid in solving one or more of the problems provided. Your should accomplish this by Scale Complexity: Try both simpler and more complicated versions of the problem to see how the approach adapts.
Your answer **MUST** be written within a Markdown isabelle code block. i.e., within a triple back ticked code block, labelled as "isabelle"

Here is some reference problems, you should evolve the skill to help solving theses problems:
{problems}

Note that you will be provided with several examples, but you should only complete the last example (i.e., the example that does not have a modified version).

############ Scale Complexity ############
{examples}

####################
Skill to evolve
“‘isabelle
{code}
“‘

Evolved skill

Figure 17: LEGO-Prover: EVOLVER “scale complexity” prompt for o3-mini.

```
theory amc12a_2021_p9
  imports Complex_Main
begin

lemma prod_sum_of_powers:
  fixes n :: nat
  shows "(\<Prod> k < n. (2^(2^k) + 3^(2^k))) = (3::nat)^(2^n) - 2^(2^n)"
proof (induction n)
  case 0
  then show ?case
    by auto
next
  case (Suc n)
  then have IH: "(\<Prod> k < n. (2^(2^k) + 3^(2^k))) =
                                                    (3::nat)^(2^n) - 2^(2^n)"
    by simp
  have "(\<Prod> k < Suc n. (2^(2^k) + 3^(2^k))) =
                  (\<Prod> k < n. (2^(2^k) + 3^(2^k))) * (2^(2^n) + 3^(2^n))"
    by (simp add: prod.atLeastLessThan_Suc)
  also have "... = ((3::nat)^(2^n) - 2^(2^n)) * (2^(2^n) + 3^(2^n))"
    using IH by simp
  also have "... = 3^(2^n) * 3^(2^n) - 2^(2^n) * 2^(2^n)"
    by (simp add: algebra_simps)
  also have "... = 3^(2^(Suc n)) - 2^(2^(Suc n))"
    by (metis power2_eq_square power_Suc power_even_eq semiring_norm(3))
  finally show ?case by simp
qed

theorem amc12a_2021_p9 :
  shows "(\<Prod> k<7. (2^(2^k) + 3^(2^k))) = (3::nat)^128 - 2^128"
proof -
  have "(\<Prod> k<7. (2^(2^k) + 3^(2^k))) =
                                  (\<Prod> k<7. (2^(2^k) + 3^(2^k)))" by simp
  then have "(\<Prod> k<7. (2^(2^k) + 3^(2^k))) = (3::nat)^(2^7) - 2^(2^7)"
    using prod_sum_of_powers[of 7] by simp
  then have "(\<Prod> k<7. (2^(2^k) + 3^(2^k))) =
                                   (3::nat)^(128) - 2^(128)" by simp
  thus ?thesis by simp
qed

end
```

Figure 18: Example gpt-4o-mini model output for `data/full_data/test/amc12a_2021_p9.json`. This is the model output prior to proof correction via manually written heuristics.

```
definition f :: "nat \<Rightarrow> nat" where
  "f x = 4^x + 6^x + 9^x"

lemma f_12k_plus_9:
  fixes k :: nat
  shows "f (12 * k + 9) = 4^(12 * k + 9) + 6^(12 * k + 9) + 9^(12 * k + 9)"
proof -
  have "f (12 * k + 9) = 4^(12 * k + 9) + 6^(12 * k + 9) + 9^(12 * k + 9)"
    by (simp add: f_def)
  thus ?thesis by simp
qed
```

Figure 19: Example lemma retrieved in the generation of the code in Figure 18. This lemma has soft use score of 0.378. It is largely irrelevant, as it is specialized to a function that does not appear in this problem.

```
lemma power_computation:
  fixes k :: nat
  shows "k^12 = (k^6)^2"
proof -
  have "k^12 = k^(6 + 6)" by simp
  also have "... = k^6 * k^6" by (simp add: power_add)
  also have "... = (k^6)^2" by (simp add: power2_eq_square)
  finally show ?thesis by simp
qed
```

Figure 20: Example lemma retrieved in the generation of the code in Figure 18. This lemma has soft use score of 0.695. While more relevant, involving powers of powers, it is again overly specialized and the proof tactics used (`simp add: power_add`, `simp add: power2_eq_square`, etc...) are not used in the PROVER's outputted solution.

```
lemma power_computation:
  fixes k :: nat
  shows "k^12 = (k^6)^2"
proof -
  have "k^12 = k^(6 + 6)" by simp
  also have "... = (k^6)^2" by (simp add: power_add)
  finally show ?thesis .
qed
```

Figure 21: Example lemma retrieved in the generation of the code in Figure 18. This lemma has soft use score of 0.706. While more relevant, involving powers of powers, it is again overly specialized and the proof tactics used (`simp add: power_add`, `simp add: power2_eq_square`, etc...) are not used in the PROVER's outputted solution.

```
lemma sum_of_powers:
  fixes n :: nat
  shows "(\<Prod> k < n. (2^(2^k) + 3^(2^k))) = (3::nat)^(2^n) - 2^(2^n)"
proof (induction n)
  case 0
  then show ?case
    by auto
next
  case (Suc n)
  then have IH: "(\<Prod> k < n. (2^(2^k) + 3^(2^k))) =
                                                (3::nat)^(2^n) - 2^(2^n)"
    by simp
  have "(\<Prod> k < Suc n. (2^(2^k) + 3^(2^k))) =
              (\<Prod> k < n. (2^(2^k) + 3^(2^k))) * (2^(2^n) + 3^(2^n))"
    by (simp add: prod.atLeastLessThan_Suc)
  also have "... = ((3::nat)^(2^n) - 2^(2^n)) * (2^(2^n) + 3^(2^n))"
    using IH by simp
  also have "... = (3^(2^n) * 3^(2^n)) - (2^(2^n) * 2^(2^n))"
    by (simp add: algebra_simps)
  also have "... = 3^(2^(Suc n)) - 2^(2^(Suc n))"
    by (metis power2_eq_square power_Suc power_even_eq semiring_norm(3))
  finally show ?case by simp
qed
```

Figure 22: Example lemma retrieved in the generation of the code in Figure 18. This lemma has soft use score of 0.96. It is reproduced almost exactly in the PROVER's output, with minor changes to parenthesization and the name of the lemma.

TroVE TabMWP (1432/5376 problems with at least one correct IMPORT Solution)	
total_cost	732
total_value	204
sum_of_values	42
range_of_numbers	10
score_difference_between	4
difference_in_medals	1
calculate_total_cost	1

Table 12: The names of functions (left column) reused in successful TroVE solutions generated by the IMPORT prompt with `CodeLlama-7b-Instruct`, and the number of solutions reusing the function (right column). Note that this is an over-estimate of the true amount of TroVE reuse, as we considered all successful IMPORT attempts, therefore assuming that TroVE's aggregator would always select the correct solution. Unlike LEGO-Prover TroVE exhibits reuse behaviour however the functions are extremely simple – trivial sums, easily implemented in `pandas`. As the baseline agent performs comparably with access to a Python compiler, it is clear that the functions in the learned library are easily retrieved from the base model's parametric knowledge.

and standard deviation is from the original paper). This observation is consistent with findings by Berlot-Attwell et al. (2024) where they found that a CREATE-SKIP ensemble had comparable performance on MATH (Hendrycks et al., 2021b) – though our findings now allow us to attribute this to the compute used by SKIP rather than ensembling as proposed by Berlot-Attwell et al. (2024).

Finally, we measured hard reuse behaviour in TroVE under TabMWP (see Table 12). We considered only reuse in correctly answered problems (optimistically considering a problem to be solved if at least one of the attempts suceeded), and removed any solutions where the tool was re-defined using the signature instead of being imported. We found that on TabMWP TroVE does exhibit reuse behaviour, but that the significantly reused learned functions are all trivial (specifically, all simple sums over a table). Given the simplicity of the learned library, it is unsurprising that the benefits vanish once the baseline has an equal computational budget.

Each experiment is performed with an RTX A6000 GPU. Approximately 550 GPU-hours were used for all experiments.

H Agent-Optimizer Evaluation

We reproduced Agent Optimizer's experimental setup and baseline (Zhang et al., 2024) using their released code[4]; the only difference is that we used `gpt-4o-2024-11-20` in place of (`gpt-4-1106-vision-preview`) as the latter is now deprecated. For reasons of cost, we limited our evaluations with `gpt-4o` to the precalculus and geometry splits that reported the largest performance increases in the original paper ($53.8 \rightarrow 62.5\%$ and $53.8 \rightarrow 58.8\%$ respectively). We also performed additional experiments with `gpt-4o-mini` on the remaining MATH splits.

As per the original paper, the library training is performed on a random training subset of 20 problems, which are then used by a ReAct-like agent tested on 80 problems. The baseline is the same ReAct-like agent without access to the library.

In general, AgentOptimizer has higher inference-time costs (see Tables 13 through 17). Unexpectedly, we also found that the baseline slightly outperformed AgentOptimizer before cost correction. This may be due to variation caused by the random train and test subsets and the small test set size (80 examples) used in the original paper – we had to resample as the authors did not release their random splits. The differences in the `gpt-4o` experiments may also be due to our use of `gpt-4o-2024-11-20` in place of the now deprecated preview. As the original paper did not use `gpt-4o-mini`, it is not unreasonable to attribute the `gpt-4o-mini` results to inferior coding or tool use skills.

Finally, we observe that AgentOptimizer exhibits true hard reuse behaviour (see Table 18). From manual inspection, the learned tools tend to be both generic and simple – as such they do not improve performance over the baseline which already has access to arbitrary Python code execution. Note that as AgentOptimizer trains the library on a separate set of problems, we consider any use of a tool in a successful solution to be an instance of reuse, as it is reuse from the train set to the test set.

Each experiment is performed with a resource allocation of 24 GB of RAM and 1 Intel Broadwell CPU cores @2.095 GHz. Approximately 34 CPU-hours were used for all experiments.

[4] https://microsoft.github.io/autogen/0.2/blog/2023/12/23/AgentOptimizer/ and https://github.com/ag2ai/ag2/blob/main/notebook/agentchat_agentoptimizer.ipynb

	4o-mini (train)	4o-mini (direct)	4o (train)	4o (direct)
Accuracy	$58.75 \pm 2.50\%$	$58.75 \pm 3.75\%$	$61.67 \pm 4.02\%$	$63.33 \pm 4.02\%$
Train	$\$0.65 \pm 0.03$	$\$0$	$\$8.73 \pm 0.47$	$\$0$
Inference	$\$0.15 \pm 0.03$	$\$0.15 \pm 0.02$	$\$2.20 \pm 0.38$	$\$1.87 \pm 0.20$

Table 13: AgentOptimizer performance on MATH precalculus. "train" indicates AgentOptimizer was used to train a library, "direct" indicates that the ReAct-like baseline was used directly without any library. We report mean and standard deviation over 3 runs.

	4o-mini (train)	4o-mini (direct)	4o (train)	4o (direct)
Accuracy	$60.00 \pm 2.50\%$	$60.00 \pm 2.50\%$	$64.17 \pm 1.44\%$	$66.25 \pm 2.50\%$
Train	$\$0.64 \pm 0.06$	$\$0$	$\$9.70 \pm 0.26$	$\$0$
Inference	$\$0.13 \pm 0.03$	$\$0.09 \pm 0.02$	$\$1.52 \pm 0.22$	$\$1.28 \pm 0.19$

Table 14: AgentOptimizer performance on MATH geometry. "train" indicates AgentOptimizer was used to train a library, "direct" indicates that the ReAct-like baseline was used directly without any library. We report mean and standard deviation over 3 runs.

	4o-mini (train)	4o-mini (direct)
Accuracy	$85.83 \pm 4.73\%$	$88.33 \pm 1.44\%$
Train	$\$0.31 \pm 0.01$	$\$0$
Inference	$\$0.13 \pm 0.01$	$\$0.07 \pm 0.01$

Table 15: AgentOptimizer performance on MATH algebra. "train" indicates AgentOptimizer was used to train a library, "direct" indicates that the ReAct-like baseline was used directly without any library. We report mean and standard deviation over 3 runs.

	4o-mini (train)	4o-mini (direct)
Accuracy	$79.17 \pm 0.72\%$	$83.75 \pm 3.31\%$
Train	$\$0.36 \pm 0.02$	$\$0$
Inference	$\$0.12 \pm 0.02$	$\$0.06 \pm 0.01$

Table 16: AgentOptimizer performance on MATH counting and probability. We report mean and standard deviation over 3 runs.

	4o-mini (train)	4o-mini (direct)
Accuracy	$75.42 \pm 4.73\%$	$79.58 \pm 2.60\%$
Train	$\$0.37 \pm 0.02$	$\$0$
Inference	$\$0.10 \pm 0.02$	$\$0.07 \pm 0.00$

Table 17: AgentOptimizer performance on MATH number theory. We report mean and standard deviation over 3 runs.

I Potential Risks

The risks of this work are low as the primary findings are negative results – ICL library learning is less effective than previously believed. This work is limited by the choice of dataset – all are in English, the performance of these systems on other languages is unstudied.

J Code, Data, and Licensing

Our code is released in the supplemental, along with the accompanying licenses. Our code is derived from TroVE (Wang et al., 2024d) (https://github.com/zorazrw/trove) released under CC-BY-SA-4.0 license, AgentOptimizer (Zhang et al., 2024) (https://github.com/ag2ai/ag2/blob/main/notebook/agentchat_agentoptimizer.ipynb) released under Apache-2.0 license, and LEGO-Prover (Wang et al., 2024a) (https://github.com/wiio12/LEGO-Prover) released under MIT license. We release our derived code under the same respective licenses.

In the course of our experiments we use the miniF2F (Zheng et al., 2021) test set of 244 examples. Specifically, we use the Isabelle version (released under Apache-2.0 license), as re-released by Wang et al. (2024a) at https://github.com/wiio12/LEGO-Prover/tree/357672c7751cd0c84aff6bf72a3d1bf97614e81d/data/full_data/test.

In our followup experiments we use the MATH dataset (Hendrycks et al., 2021b) available under MIT license. The AgentOptimizer uses randomly selected subsets of 20 train examples and 80 test

MATH Precalculus Run 0 (53/80 Successes)	
Reused Function	Reuses
evaluate_math_expressions	12
solve_math_problem	9
evaluate_trig_expressions	7
evaluate_inverse_trig_expressions	3
evaluate_matrix_expression	1
solve_trigonometric_identity	1
evaluate_cross_product	1
MATH Precalculus Run 1 (47/80 Successes)	
solve_question_with_sympy	8
evaluate_math_expression	6
compute_matrix_multiplication	1
compute_vector_projection	1
MATH Precalculus Run 2 (48/80 Successes)	
execute_math_code	21
transform_and_solve	11
evaluate_expression	5
simplify_trig_expression	2
MATH geometry Run 0 (52/80 Successes)	
evaluate_expression	2
calculate_distance_2d	1
calculate_triangle_area	1
MATH geometry Run 1 (50/80 Successes)	
solve_math_problem_fixed	10
solve_math_problem	6
evaluate_math_problem	4
solve_task_with_sympy	3
calculate_area_and_volume_of_cone	1
calculate_distance_3d	1
MATH geometry Run 2 (52/80 Successes)	
evaluate_expression	8
evaluate_algebra	4
evaluate_trigonometric	3

Table 18: The names of functions (left column) reused in successful AgentOptimizer solutions with gpt-4o, as well as the number of solutions reusing the function (right column). Note that as AgentOptimizer uses tools developed in a training phase, we consider any use in a successful test time solution to be reuse. While AgentOptimizer exhibits significant reuse behaviour, unlike LEGO-Prover, the learned functions tend to be extremely generic – typically chaining sympy.sympify with sympy.simplify. Given the baseline agent performs comparably with access to a Python compiler, it is clear that the functions in the learned library are easily retrieved from the base model's parametric knowledge.

examples – we release the exact questions used in our supplemental material. TroVE uses the full MATH test set (881 algebra problems, 291 counting and probability problems, 237 geometry problems, 503 intermediate algebra problems, 497 number theory problems, 636 prealgebra problems, and 156 precalcus problems), we use the version re-released by Wang et al. (2024d) at https://github.com/zorazrw/trove/tree/c4d16b6a2e38020540db2a611fdea722da6b880c/data/math.

Our follow experiments also use the 5376 problem test split of the TabMWP dataset (Lu et al., 2023) available under MIT license. We again use the version re-released by Wang et al. (2024d) (https://github.com/zorazrw/trove/tree/c4d16b6a2e38020540db2a611fdea722da6b880c/data/tableqa).

All datasets are English datasets of mathematical problems. The formal components of the miniF2F split are in the ISAR language used by the Isabelle theorem prover. TabMWP contains tables in markdown format.

All code & data is used in a manner consistent with the terms of release, and the authors' intent (i.e., for scientific research of AI mathematical reasoning).

K Use of AI Assistants

In the creation of this research, ChatGPT was used in a minimal capacity. Specifically, it was used for the generation of some TroVE code fragments, to find 2 additional pieces of related work during the initial literature review, and to suggest revisions to the initial human-written manuscript. All AI outputs were carefully reviewed for correctness.