Is This LLM Library Learning? Evaluation Must Account For Compute and Behaviour

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., storing generated tools) – all without fine-tuning. However, we find strong reasons to be skeptical. Specifically, we identify a serious evaluation flaw present in a large number of ICL library learning works: these works *do not correct for the difference in computational cost* between baseline and library learning systems. Studying three separately published ICL library learning systems, we find that all of them fail to consistently outperform the simple baseline of prompting the model – improvements in task accuracy often vanish or reverse once computational cost is accounted for. Furthermore, 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).

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. An equal computational budget must be used for baselines, alongside behavioural analysis.[1]

1 Introduction

Humans extend their capabilities, improve their reliability, and save mental effort by 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) writes code for 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 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. 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. (2024d) and Yuan et al. (2024) hypothesize that library learning encourages simpler and more reliable functions. Similarly, Qian et al. (2023) suggest that performance improves through separating the task into abstract reasoning and instantiation tasks (i.e., creating a generic tool before calling it with the problem's specifics). Wang et al. (2024b) propose that their system accumulates reusable relevant reference examples. These systems may also claim to reduce cost by caching the created tools (Cai et al., 2024).

In this work, we investigate three such ICL library learning systems: LEGO-Prover (Wang et al., 2024b), TroVE (Wang et al., 2024e), and AgentOptimizer (Zhang et al., 2024). TroVE and AgentOptimizer generate Python code for solving math word problems, generating a library of functions and callable tools respectively. LEGO-Prover autoformalizes theorems, i.e., converts natural language proofs into formal proofs verified by the Isabelle theorem prover (Paulson, 1994). Wang et al. (2024b) reported SotA performance at the

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

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 tackling increasingly intricate mathematical problems." Based on these and similar claims, we answer two important questions: 1) Do these systems exhibit library learning behaviours? 2) Does this behaviour improve performance?

Our contributions are as follows: (1) We evaluate direct library reuse behaviour in LEGO-Prover (Wang et al., 2024b), TroVE (Wang et al., 2024e), and AgentOptimizer (Zhang et al., 2024). We find flawed reuse behaviour in two systems: TroVE tends to reuse trivial functions, and LEGO-Prover's library is not directly reused. (2) We provide evidence that LEGO-Prover's learned library does not serve as a useful reference for solving related tasks. (3) Critically, we demonstrate that all three systems fail to consistently outperform a simple repeated prompting baseline once the baseline is provided with an equivalent computational budget. Future works must account for this in their evaluations. Together, these three points suggests that a serious re-examination of ICL LLM-based library learning is in order. (4) Based on our findings, we make concrete proposals for improving the development and evaluation of library learning systems.

2 Background

We begin with a brief overview of the purpose and design of the three library learning systems studied. LEGO-Prover (i.e., LP) uses in-context learning (ICL) to convert natural language proofs into verifiable formal proofs via two pools of concurrent processes that share a library: the PROVER and EVOLVER. The PROVER iterates over unconverted proofs: each natural language proof is first used to generate requests for potentially useful lemmas, and then reworded into an English step-by-step proof. The rephrased proof is augmented with lemmas retrieved from the learned library, converted into a formal proof, processed by corrective heuristics, and finally verified or rejected by Isabelle. Simultaneously, the EVOLVER proves and revises requested lemmas, adding these to the library. The PROVER cycles through unsolved problems; the maximum number of attempts per problem is a hyperparameter. As per Wang et al. (2024b), we use Draft-Sketch-Prove (Jiang et al., 2023) as a baseline. Draft-Sketch-Prove (i.e., DSP) uses ICL and post-processing heuristics to convert natural language into an Isabelle proof. We modify DSP to use LEGO-Prover's heuristic post-processor.

AgentOptimizer (Zhang et al., 2024) and TroVE (Wang et al., 2024e) solve math word problems via Python code generation. TroVE is an online method that generates $3k$ independent solutions for each problem. The first k directly prompt the LLM for a solution (i.e., SKIP solutions). The next k condition on the library (i.e., IMPORT solutions). The final k generate and use a new function for the library (i.e., CREATE solutions). $k = 5$ is used in the paper. Majority vote selects the answer; heuristics add & remove functions from the library. As per Wang et al. (2024e), we use direct prompting (i.e., SKIP) with majority vote as the baseline. AgentOptimizer creates a library of Python functions that are exposed to an LLM at inference time via ICL tool calling. Learning is via offline training: ICL repeatedly adds, revises, or deletes library items to optimize LLM performance on a train set. As per Zhang et al. (2024), ReAct (Yao et al., 2023) with a Python interpreter tool is used as the baseline.

3 Reuse Behaviour of Several Systems Under Various LLMs

ICL library learning has reported significant improvements over baselines: TroVE reports 11% on TabMWP ($36\% \rightarrow 47\%$), AgentOptimizer reports 8.7% on the MATH Precalulus split ($53.8\% \rightarrow 62.5\%$), and LEGO-Prover reports 10.7% on miniF2F ($39.3\% \rightarrow 50\%$). What drives these gains? The most obvious explanation is the development of a library of useful tools. These may be broadly applicable tools that can be **directly reused** in the problem (e.g., a mathematician applying Rolle's theorem or a programmer calling `numpy.std`). Alternatively, the library may instead be useful exemplars that are **softly reused** (i.e., fragments reproduced with minor edits). This behaviour is akin to forking a code repository as the starting point for a new project. Note that genuine library learning requires **reuse**, rather than (single) **use**. A bespoke tool, useful for only a single specific query, does not belong in a library. Indeed, creating bespoke tools is a form test-time scaling rather than library learning.

In search of the source of performance gains, we run these three systems and study their behaviour.

3.1 Do TroVE and AgentOptimizer Exhibit Reuse?

As per the original paper, we evaluate TroVE on math word problems with MATH (Hendrycks et al., 2021b), and tabular reasoning with TabMWP (Lu et al., 2023), HiTab (Cheng et al., 2022), WTQ (Pasupat and Liang, 2015). We use `CodeLlama-7B-Instruct` as per the original and also study `Llama-3.1-8B-Instruct`. We follow AgentOptimizer's evaluation protocol on MATH, randomly sampling the train set. As in the original work we use GPT models (`4o-mini` due to the deprecation of GPT-3.5 & GPT-4-preview. We also run `4o` on the MATH geometry and precalculus splits).

To measure direct reuse, we consider only correctly answered problems – reuse in wrong answers cannot explain performance gains. TroVE generates IMPORT solutions via ICL on the source code of tools, generating code that is executed in a Python environment that already has the tools defined. For TroVE, we optimistically consider a problem to be solved if at least one IMPORT solution succeeded (thereby assuming that this answer will be selected by majority vote). We removed any solutions where the tool was re-implemented with the signature instead of being imported and called (i.e., soft reuse). Tool-use in AgentOptimizer is simpler as it uses the OpenAI tool-calling API; we consider any use of a tool in a correct solution to be reuse (as the tool is reused from the train set).

We find that TroVE and AgentOptimizer both exhibit direct reuse behaviour; see Tab. 1 for direct function reuse in TabMWP and MATH respectively, and App. G for the full MATH & TabMWP reuse tables. We find direct reuse under all LLMs tested. However, unlike AgentOptimizer, TroVE does not reliably learn functions. Over the three runs and two LLMs no functions were learned on the MATH geometry & intermediate algebra splits, and only a single function was learned on the precalculus split. Furthermore, the learned functions are largely trivial: see Fig. 1 for three randomly sampled examples, and contrast with a random sample from AgentOptimizer in Fig. 2. Summarizing, both systems exhibit direct reuse, though TroVE's performance gains are curious given the simpler library and unreliable function learning.

TroVE (CodeLlama)	
TabMPW: 4001/80640 correct IMPORT attempts	
`total_cost (seed 2)`	2716
`median_of_column`	1197
`total_cost`	1148
`sum_of_values`	652
`mode (seed 3)`	625
`sum_column`	440
18 Functions redacted for space	...
`range_of_values`	5
TroVE (Llama3.1):	
TabMWP: 5425/80640 correct IMPORT attempts	
`sum_column`	2326
`rate_of_change`	2230
`mode`	1447
`median`	1442
`mean_value`	1381
24 Functions redacted for space	...
`get_total_cost`	2
AgentOptimizer (gpt-4o)	
MATH geometry: 154/240 correct solution attempts	
`solve_math_problem_fixed`	10
`evaluate_expression` [Run 3]	8
`solve_math_problem`	6
`evaluate_math_problem`	4
`evaluate_algebra`	4
`evaluate_trigonometric`	3
`solve_task_with_sympy`	3
`evaluate_expression` [Run 1]	2
`calculate_distance_2d`	1
`calculate_triangle_area`	1
`calculate_area_and_volume_of_cone`	1
`calculate_distance_3d`	1

Table 1: The functions (left column) reused in successful AgentOptimizer & TroVE solutions, and the number of reuses (right column). There is reuse behaviour, however the learned functions tend to be simple. We aggregate over 3 runs. See App. G.1 for the full TroVE table, and equivalent tables on all 7 MATH splits, HiTab, and WTQ. See App. G.3 for all AgentOptimizer tables.

3.2 Does LEGO-Prover Exhibit Reuse?

As per the original paper, we evaluate LEGO-Prover on miniF2F, though replacing GPT-3.5 with `4o-mini` due to deprecation. We also evaluate `LLama 3.1-8B-Instruct`, and run `o3-mini`, `Qwen3-14B`, and `4o` on subsets of the data.

We define a lemma 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). We test for direct reuse via the methodology by Berlot-Attwell et al. (2024): for every retrieved lemma provided to the PROVER we check for the presence of the lemma (verbatim use) or the lemma's name (name use) in the

Model	% of miniF2F Test Set	Successful Attempts	Lemmas in Prompts	Lemma Use		Lemma Reuse			
				Verbatim	Name	Verbatim		Name	
						1	2+	1	2+
o3-mini	10%	20/1825	75	0	5	0	0	0	0
Qwen3-14B-Instr	10%	22/2741	83	5	54	0	0	0	0
GPT-4o	20%	43/5537	121	2	9	0	0	0	0
Llama3.1-8B-Instr	100%	212/28679	721	56	231	1	0	2	1
4o-mini	100%	189/57419	583	13	47	0	0	1	0

Table 2: Lemma use and reuse in successful PROVER attempts. We report the proportion of successful PROVER attempts and the number of unique retrieved lemmas in the PROVER's input prompts. We also report lemma use (i.e., the lemma appears in whole or by name in at least one solution) and reuse (i.e., use in several solutions). We sum over 3 runs. Virtually zero reuse is observed in our data, or the original Wang et al. (2024b) logs (App. Tab. 10).

```
def number_of_groups(n, k):
  return n // k

def is_multiple_of_12(num):
  if num % 12 == 0:
    return True
  else:
    return False

def is_cube(x):
  return x ** 3 == x
```

Figure 1: Three functions randomly sampled from TroVE's final libraries. Note the simplicity and errors.

```
def calculate_cone_properties(radius, height):
 volume = (1/3) * math.pi * radius**2 * height
 slant_height = math.sqrt(radius**2 + height**2)
 lat_surf_area = math.pi * radius * slant_height
 return {'volume': volume,
         'lateral_surface_area': lat_surf_area}
```

Figure 2: Function randomly sampled from AgentOptimizer's final library; unlike TroVE, AgentOptimizer consistently learns functions on the MATH geometry split. `lat_surf_area` was renamed for brevity.

PROVER's output. We find evidence for frequent direct use, *but not reuse*, of lemmas (see Tab. 2). I.e., LEGO-Prover can use relevant lemmas in its proofs, but the lemmas it learns are only directly usable in a single problem.

While this bodes poorly, the original authors claim that "[m]any skills [...] are very helpful as reference examples". To test this, we quantify the degree of 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

```
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)
  ...
```

Figure 3: Sample LEGO-Prover lemma that with a high soft use score of 0.96, reproduced almost exactly in the final proof (App. Fig. 27). Despite being used, reuse in other tasks is extremely challenging given that $\prod_{k<n}(2^{2^k}+3^{2^k}) = 3^{2^n} - 2^{2^n}$ is a specialized lemma.

then $\text{soft_use}(L,P) := 1 - \frac{|\text{LCS}(L,P)|}{|L|}$ where LCS is the longest common subsequence. Note the proof is not penalized for additional tokens as the final proof *should* add to the used lemma. See App. F for example lemmas and their soft use scores.

Again we find evidence for use, but not reuse. Lemmas such as Fig. 3 are used heavily in a single proof, but are too specialized for broader usefulness. To better illustrate this, we use the soft-use metric to plot two survival curves, i.e. the proportion of solved tasks exhibiting soft use, or soft reuse, at a given threshold (see Fig. 4). For a given threshold τ of the metric, we say a task with proof P exhibits soft use if it has a retrieved lemma L such that $\text{soft_use}(L,P) \geq \tau$. Such a task also exhibits soft reuse if at least one of these lemmas is soft-used by another task. 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. These curves also reveal an upper bound on the proportion of tasks that can benefit from lemma reuse: this bound arises from the *retriever*

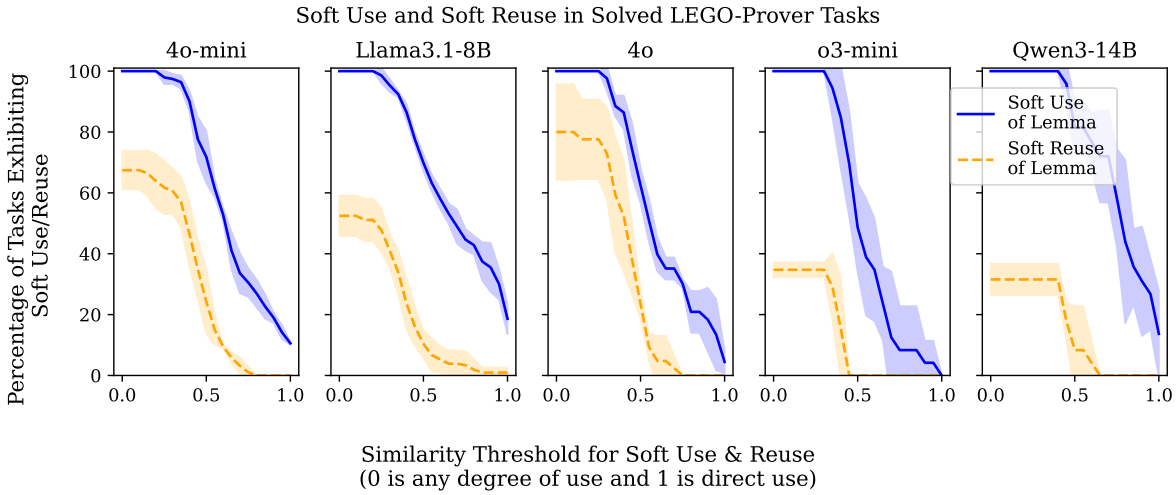

Figure 4: We define a normalized score for the degree to which a lemma is used in a proof (i.e., soft use). Using this score and a threshold, we can determine whether LEGO-Prover's solution to a task soft-uses the retrieved lemmas provided in the PROVER's context. In blue we plot the percentage of solutions meeting the desired threshold for soft use. In orange, we plot the percentage of solutions meeting soft-reuse (i.e., soft using a lemma that is also soft-used by a different task). We observe that the lines for reuse rapidly reach zero for even moderate similarity thresholds (e.g., $\tau = 0.6$) whereas soft 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.

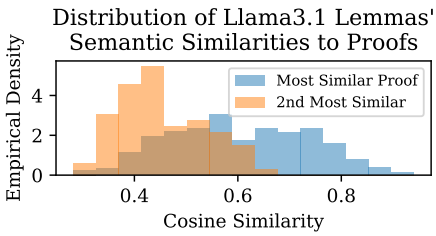

Figure 5: We plot the distribution over lemmas of the semantic similarity with the most similar proof they were retrieved for, and the second most similar proof they were retrieved for. Reused lemmas would be similar to both, but we instead find a sharp decrease in similarity. See App. C.1.1 for the plots under different LLMs.

that selects lemmas from the library. Some tasks do not retrieve any lemmas that are also retrieved by another task and therefore cannot exhibit PROVER reuse. Consequently, at the threshold $\tau = 0$ (i.e., all retrieved lemmas are soft used) the survival curve for reuse attains less than 100%. In App. C.1 we compare the soft use scores of retrieved, non-retrieved, and unrelated Isabelle lemmas – again finding no evidence for soft reuse.

At this point we have eliminated the possibility of soft reuse that preserves syntax. Weaker forms of semantic reuse are possible, but fraught as they deviate further from the verified lemma. To test for such reuse, we calculate the semantic similarity of lemmas and proofs using OpenAI `text-embedding-3-large` embeddings. If semantic reuse were occurring, we would expect there to exist lemmas highly similar to two proofs. Instead, we see a sharp drop in similarity from the most to second most similar proof – see Fig 5.

4 What drives performance gains?

These results are curious as, by following the authors' evaluation methodologies, we often see very large performance gains: e.g., our experiments find a 8.4% miniF2F subset improvement in LEGO-Prover under o3-mini (19.4% $\rightarrow$ 27.8%), and a 6.6% improvement on MATH algebra with TroVE under CodeLlama (21.9% $\rightarrow$ 28.5%). Yet TroVE's library is simple and LEGO-Prover's is not reused at all. Rather than library learning, we propose that increased compute is the key factor.

Simply put, library learning adds a computational overhead that current evaluation methodologies fail to capture. Typically, a library learning system is compared to a baseline, matched by iterations. This is how LEGO-Prover was evaluated, and it is a standard practice seen in many ICL library learning works; e.g., Voyager (Wang et al., 2024a) and Dynasaur (Nguyen et al., 2025). These works compare systems based on the code revisions and agent steps respectively, not considering differences in the cost of said steps. A special case

LLM	System	miniF2F	Avg. Cost
gpt-4o†	LP	$29.3_{\pm 3.1}\%$	\$125.4
	DSP	$\mathbf{23.1_{\pm 1.2}}\%$	\$22.9
	LP (CC↓)	$\mathbf{23.8_{\pm 1.2}}\%$	\$22.6
o3-mini‡	LP	$27.8_{\pm 4.8}\%$	\$218.0
	DSP	$\mathbf{19.4_{\pm 2.4}}\%$	\$16.5
	LP (CC↓)	$15.3_{\pm 2.4}\%$	\$17.4
Qwen3-14B‡	LP	$30.6_{\pm 2.4}\%$	109.0M
	DSP	$\mathbf{20.8_{\pm 4.2}}\%$	7.0M
	LP (CC↓)	$15.3_{\pm 2.4}\%$	6.5M
4o-mini*	LP	$25.8_{\pm 1.6}\%$	\$69.9
	DSP	$\mathbf{35.9_{\pm 0.9}}\%$	\$12.1
	LP (CC↓)	$17.5_{\pm 0.9}\%$	\$11.9
Llama3.1*	LP	$29.0_{\pm 2.1}\%$	242.9M
	DSP	$8.2_{\pm 0.4}\%$	40.5M
	LP (CC↓)	$\mathbf{15.7_{\pm 1.0}}\%$	38.9M

Table 3: Compute-Corrected LEGO-Prover.

LLM	System	MATH Geo.	Avg. Cost
4o	AgentOpt.	$\mathbf{64.2_{\pm 1.4}}\%$	\$11.22
	ReAct	$\mathbf{66.3_{\pm 2.5}}\%$	\$1.28
4o-mini	AgentOpt.	$\mathbf{60.0_{\pm 2.5}}\%$	\$0.77
	ReAct	$\mathbf{60.0_{\pm 2.5}}\%$	\$0.09

LLM	System	MATH Precalc.	Avg. Cost
4o	AgentOpt.	$\mathbf{61.7_{\pm 4.0}}\%$	\$10.94
	ReAct	$\mathbf{63.3_{\pm 4.0}}\%$	\$1.87
4o-mini	AgentOpt.	$\mathbf{58.75_{\pm 2.5}}\%$	\$0.81
	ReAct	$\mathbf{58.8_{\pm 3.8}}\%$	\$0.15

Table 4: AgentOptimizer performance and computational costs. Following Zhang et al. (2024) we train and evaluate on random subsets of the full MATH splits. See App. G.3 for `4o-mini` results on additional MATH splits.

		MATH							TableQA		
LLM	System	Algebra	Counting	Geometry	Intermediate	Number	Prealgebra	Precalculus	TabMPW	WTQ	HiTab
CodeLlama 7B-Instr.	Direct	$21.8_{\pm 1.7}\%$	$21.2_{\pm 1.3}\%$	$7.4_{\pm 1.2}\%$	$11.6_{\pm 1.1}\%$	$25.9_{\pm 1.2}\%$	$29.1_{\pm 2.3}\%$	$12.3_{\pm 0.9}\%$	$44.3_{\pm 0.5}\%$	$20.0_{\pm 0.4}\%$	$14.8_{\pm 0.3}\%$
	TroVE	$\mathbf{29.1_{\pm 0.7}}\%$	$\mathbf{26.9_{\pm 2.2}}\%$	$7.7_{\pm 1.1}\%$	$\mathbf{12.9_{\pm 1.0}}\%$	$\mathbf{29.6_{\pm 0.9}}\%$	$\mathbf{32.3_{\pm 2.1}}\%$	$\mathbf{20.4_{\pm 1.5}}\%$	$40.1_{\pm 2.6}\%$	$\mathbf{21.0}\%$	$\mathbf{18.0}\%$
	Direct (CC↑)	$26.7_{\pm 0.5}\%$	$23.6_{\pm 0.4}\%$	$\mathbf{8.4_{\pm 1.2}}\%$	$\mathbf{13.6_{\pm 0.9}}\%$	$28.0_{\pm 1.8}\%$	$\mathbf{32.8_{\pm 1.8}}\%$	$15.4_{\pm 0.4}\%$	$\mathbf{45.4_{\pm 0.4}}\%$	$18.7_{\pm 0.3}\%$	$17.1_{\pm 0.4}\%$
Llama3.1 8B Instr.	Direct	$34.6_{\pm 0.2}\%$	$29.1_{\pm 0.7}\%$	$12.9_{\pm 1.1}\%$	$19.1_{\pm 0.2}\%$	$35.6_{\pm 0.4}\%$	$43.2_{\pm 0.6}\%$	$15.8_{\pm 1.7}\%$	$64.2_{\pm 0.6}\%$	$20.7_{\pm 0.2}\%$	$19.5_{\pm 0.6}\%$
	TroVE	$36.3_{\pm 1.1}\%$	$\mathbf{28.9_{\pm 3.9}}\%$	$14.1_{\pm 1.4}\%$	$19.8_{\pm 1.5}\%$	$\mathbf{38.2_{\pm 2.1}}\%$	$40.8_{\pm 4.9}\%$	$\mathbf{20.1_{\pm 2.0}}\%$	$57.2_{\pm 0.9}\%$	$14.3_{\pm 0.8}\%$	$22.3_{\pm 0.4}\%$
	Direct (CC↑)	$\mathbf{40.1_{\pm 0.7}}\%$	$\mathbf{30.9_{\pm 0.7}}\%$	$\mathbf{16.3_{\pm 1.4}}\%$	$\mathbf{22.5_{\pm 0.5}}\%$	$\mathbf{40.1_{\pm 0.6}}\%$	$\mathbf{49.0_{\pm 0.6}}\%$	$\mathbf{19.2_{\pm 1.0}}\%$	$\mathbf{64.1_{\pm 0.1}}\%$	$\mathbf{19.4_{\pm 0.1}}\%$	$\mathbf{23.5_{\pm 0.5}}\%$

Table 5: Compute-Corrected TroVE. Results aggregated over five seeds for CodeLlama and three for Llama3.1.

Figure 6: Compute-Corrected experiment results, over 3 trials unless otherwise indicated. Library learning systems use more compute than baselines; after Compute-Correction (CC in tables) performance is generally comparable. Uncorrected rows are coloured in grey. We bold the higher performing system under comparable compute, bolding both if within a standard deviation. Note that *, †, and ‡ indicate 100%, 20%, and 10% of the miniF2F dataset.

of this evaluation approach is where the number of iterations is fixed to 1, i.e., comparing performance under a single attempt. TroVE & AgentOptimizer do this, disadvantaging the baselines by not considering the cost of creating and ingesting the library.

Specifically, comparing Draft-Sketch-Prove and LEGO-Prover on a fixed number of PROVER attempts is problematic, as it 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. The PROVER is also more computationally intensive, as it performs a 2-stage process of decomposition and formalization. Similarly, TroVE uses a SKIP-only ablation with $k = 5$ as their baseline, without accounting that TroVE samples $3k$ generations.

4.1 Matching System Compute

To place our baselines on an equal footing, we approximately equalize compute by matching a weighted sum of input and output tokens. For proprietary models we use input and output token costs as the weighting, for open source models we use a 1:1 weighting (i.e., total tokens). Using this proxy, we find LEGO-Prover requires 6-14 times more compute per iteration than Draft-Sketch-Prove (see Tab. 3). We compensated by decreasing LEGO-Proverś iterations to approximately match.

In preliminary experiments we found that TroVE used approximately 3 three times as much compute as its baseline, and found that we could approximately compute-match by running the SKIP ablation with 15 samples. On a subset of TabMPW we found that the compute-matched ablation uses 87.1% as many tokens on average as full TroVE.

We find AgentOptimizer uses 5-8 times more compute once the training stage is considered (see Tab. 4). If we disregard the training cost as amortizable, we still find that inference costs typically increase by 17%-40% due to the overhead of ingesting the library (See App. G.3). One may correct for this by performing best-of-N sampling when the gap is large, and otherwise increasing the number of ReAct steps. Ultimately neither experiment was necessary for reasons outlined in the next section.

4.2 Compute-Matched Performance

In LEGO-Prover and TroVE we find that the performance gap typically vanishes once the baseline is provided a comparable computational budget. Unexpectedly, despite having the best library learn-

ing behaviour, we find that AgentOptimizer exhibits no improvement over the baseline under the original methodology while still incurring higher costs. These findings are summarized in Fig. 6. Additional tables with AgentOptimizer's performance and cost on the remaining MATH splits under `gpt-4o-mini` can be found in App. G.3.

In a followup experiment we consider the possibility that Draft-Sketch-Prove plateaus before LEGO-Prover, or that LEGO-Prover requires enough iterations for the library to reach a critical mass. To account for this we instead increased the number of Draft-Sketch-Prove iterations to approximately match LEGO-Prover compute. Doing so, we find that Draft-Sketch-Prove typically remains within one standard deviation or outperforms LEGO-Prover throughout the evaluation (see Fig. 7). Llama3.1 is the exception, however the plot makes clear that Draft-Sketch-Prove improves with additional compute, and over half of the gap is attributable to a different in computational budget.

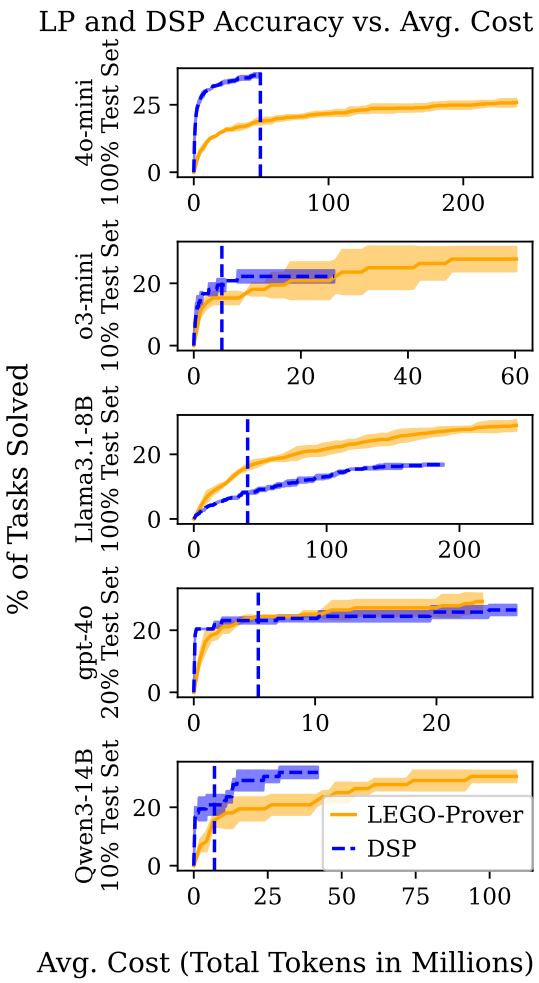

Figure 7: Average LEGO-Prover and Draft-Sketch-Prove performance compared by token usage in millions. The shaded region is 1 standard deviation (3 trials). Draft-Sketch-Prove generally has comparable or superior performance. The vertical line is the number of Draft-Sketch-Prove prover attempts equal to LEGO-Prover's. Due to cost, o3-mini and Qwen3 were run on a random 10% test subset; gpt-4o was run on 20% subset. As o3-mini cost over twice our predicted cost, we were unable to run Draft-Sketch-Prove for a comparable budget. LEGO-Prover Llama3.1 illustrates the importance of matching compute, as Draft-Sketch-Prove performance continues to improve with compute: over half of the gap when matching iterations is attributable to unequal computational budget.

5 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. 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. Berlot-Attwell et al. (2024) reported low direct reuse in LEGO-Prover and TroVE, and propose that the performance gains are due to self-correction, ensembling, or indirect reuse. 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 reasoning models) (2) advances our understanding by providing new evidence against the soft reuse hypothesis, and (3) provides a simple compute-based 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 conditioned on a human provided Python library for vision pro-

cessing), 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 tools and would perform better with 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 often fail 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 the techniques studied are of mixed efficacy at best and fail to operate as believed, suggesting that there are fundamental flaws in these approaches.

6 Conclusions

Our work combines with other emerging evidence to suggest serious flaws in the assessment of ICL LLM 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; LATM (Cai et al., 2024) is closest, discussing asymptotic complexity, but that cannot control for the effects outlined in this paper.

Consequently, the observed benefits of library learning may be largely or wholly 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 their evaluation fixed all models to 20 steps. I.e., it did not account for the differences in computational cost per step between DynaSaur, directly prompting gpt-4o, Sibyl (Wang et al., 2024c), 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). Caching generated tools is moot if said tools are not productively and cheaply reused, and our analysis suggests that they are not – at least in the systems studied. The savings from caching must exceed the cost of creating and selecting these tools – we perform the first cost-controlled comparison of LLM library learners that we are aware of and, in most cases, 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 questions the reliability and efficacy 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, and potentially combined with curriculum learning). 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.

7 Limitations

For reasons of cost, `o3-mini`, `gpt-4o` and `Qwen3-14B` were run on random subsets of miniF2F. Despite this, we observe the same LEGO-Prover trends across most of the 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.

Fig. 7 is plotted using the average cost per attempt (see App. 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. Having said this, we expect that our experiments with semantic similarity would capture such reuse.

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 8 & 9 that while Draft-Sketch-Prove reproduces Wang et al. (2024b)'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. (2024b) (see App. Tab. 10).

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 App. C.2) we believe that minor rephrasing of our final prompts would not change our results.

While imbalanced computational budget explains the majority of our results, it cannot wholly explain why LEGO-Prover outperforms Draft-Sketch-Prove under Llama3.1; though we can say with confidence that LEGO-Prover is not performing library learning. We speculate that Llama3.1 is more effective at code-refinement than zero-shot generation, but we leave confirming or disproving this for future work.

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.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

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.

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. 2025. Dynasaur: Large language agents beyond predefined actions. In *Second Conference on Language Modeling*.

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.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

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.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 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. 2024b. 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. 2024c. Sibyl: Simple yet effective agent framework for complex real-world reasoning. *CoRR*, abs/2407.10718.

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, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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, Llama3.1-8B-Instruct, and Qwen3-14B-Instruct. As per the original paper, retrieval is performed via Ada embeddings (specifically, text-embedding-ada-002).

LEGO-Prover's hyperparameters are listed in Tab. 6; values changed from defaults are underlined. Note that Wang et al. (2024b) use 50 attempts for their ablation experiments. Draft-Sketch-Prove's hyperparameters are listed in Tab. 7. The prompts used in our experiment are documented in App. D. All our experiments use Isabelle 2022.

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. Additionally, 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 in the same manner as LEGO-Prover. I.e., once a problem is solved we cease generating proofs for this task. Finally, note that DSP's final step during operation is to apply post-processing with heuristics to fill or correct gaps in the proof. In our experiments we ensure fair coparison by modifying DSP to use LEGO-Prover's post-processor and wrapper of the Isabelle

LEGO-Prover Hyperparameter	Value
Temperature	0.7, (N/A for o3-mini)
# PROVER Processes	3
# EVOLVER Processes	8
# Attempts	100 for 4o-mini on 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 6: LEGO-Prover hyperparameters. Values changed from Wang et al. (2024b) 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 7: Draft-Sketch-Prove hyperparameters. Values changed from Jiang et al. (2023) are underlined.

verifier, increasing the timeout of both systems to 120s as used by Draft-Sketch-ProveṄote that LEGO-Prover's wrapper was derived from that of Draft-Sketch-Prove and it applies a superset of the original Draft-Sketch-Prove heuristics for correcting generated proofs.

Each experiment with proprietary models was 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 these experiments. The experiments with open source models were performed with 200GB of RAM, 50 Intel Xeon Gold 6338 cores, and one NVIDIA L40S. Approximately 30 GPU-days were used over the course of these 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 unrelated Isabelle lemmas used to generate Fig. 8 are extracted from the `2019-08-19` Archive of Formal Proofs[2]. 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 LEGO-Prover Details and Additional Experiments

We evaluated LEGO-Prover and Draft-Sketch-Prove on the miniF2F (Zheng et al., 2021) test set; see Tab. 8 for results with gpt models before cost-correction. In this setting, we follow Wang et al. (2024b) and 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.

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., 2024b). 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. (2024b). This makes the discrepancy with Wang et al. (2024b)'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, find-

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

ing no such evidence (see App. C.2).

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

C.1 Additional LEGO-Prover Soft Use and Reuse Experiments

In this section we outline further evidence of soft use in LEGO-Prover. 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 baseline 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)[3].

These survival curves are plotted in the top row of Fig. 8. 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.

Therefore, to study LEGO-Prover's soft reuse behaviour, we repeat our survival curves but instead plot the proportion of lemmas used in at least two solutions generated by the LLM (see Fig. 8, 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.

C.1.1 Additional Density Figures

We plot the distribution over lemmas of the semantic similarities of the most and second most similar lemmas for each LLM tested in LEGO-Prover.

Fig. 5 of the main text reports the distribution for Llama3.1; here Fig. 9 through Fig. 12 report the distributions for `4o`, `4o-mini`, `o3-mini`, and `Qwen3-14B`.

C.2 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 App. 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 the methodology outlined 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 prompt). See Tab. 9 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 13 and 14.

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 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.

[3] `https://www.isa-afp.org/`; see App. B for details.

	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 8: Evaluation of LEGO-Prover and Draft-Sketch-Prove for a fixed number of prover attempts on various subsets of the miniF2F test set. Note that this follows the original evaluation methodology, therefore failing to account for differences in computational cost. Due to 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.

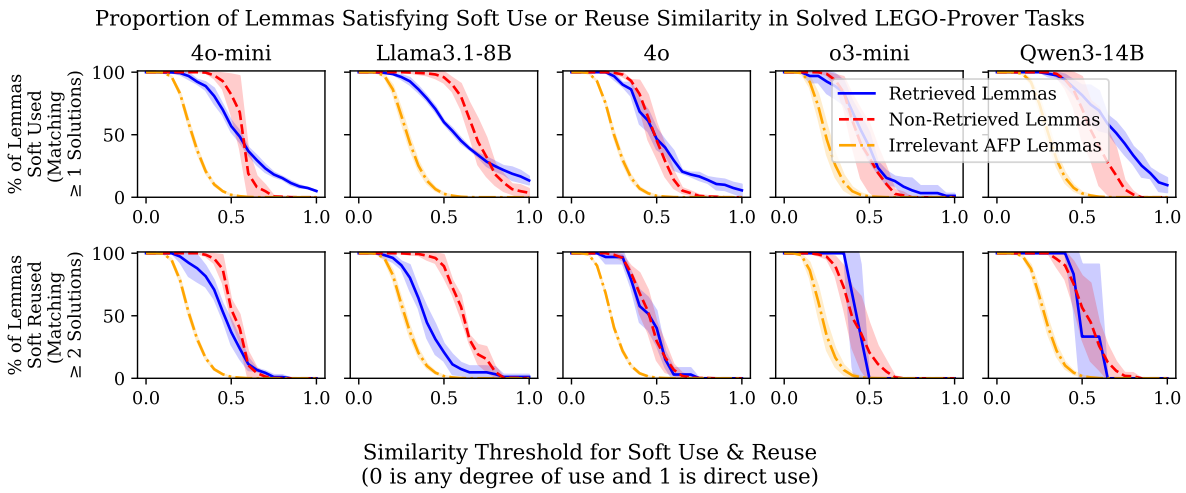

Figure 8: These plots demonstrate that LEGO-Prover exhibits soft use, but not reuse, of lemmas. The top row concerns use. It plots the percentage of lemmas meeting a given the threshold for soft use in at least one solution (see Section 3.2 for the definition of this score). The lemmas available to the 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 3.2 for a detailed explanation.

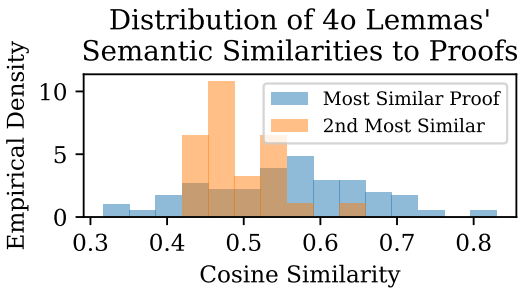

Figure 9: For LEGO-Prover under `gpt-4o` we plot the distribution over lemmas of the semantic similarity with the most similar proof they were retrieved for, and the second most similar proof they were retrieved for. Reused lemmas would be similar to both. Instead, we instead find a sharp decrease in similarity.

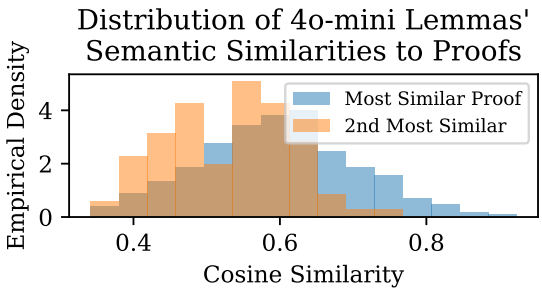

Figure 10: For LEGO-Prover under `gpt-4o-mini` we plot the distribution over lemmas of the semantic similarity with the most similar proof they were retrieved for, and the second most similar proof they were retrieved for. Reused lemmas would be similar to both. Instead, we instead find a sharp decrease in similarity.

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 9: 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.

				Verbatim reused		Name reused	
Model	% of Test Set	Successful Attempts	Lemmas in Prompts	1	2+	1	2+
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 10: This table is analogous to Tab. 2 (reuse in successful PROVER attempts), however the reuse numbers are calculated from the original LEGO-Prover logs released by Wang et al. (2024b). Again, we find negligible reuse.

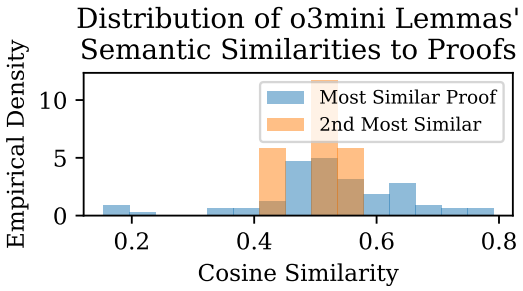

Figure 11: For LEGO-Prover under `o3-mini` we plot the distribution over lemmas of the semantic similarity with the most similar proof they were retrieved for, and the second most similar proof they were retrieved for. Reused lemmas would be similar to both. Instead, we instead find a sharp decrease in similarity.

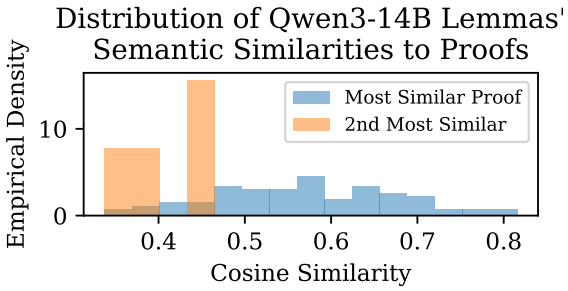

Figure 12: For LEGO-Prover under `Qwen3-14B` we plot the distribution over lemmas of the semantic similarity with the most similar proof they were retrieved for, and the second most similar proof they were retrieved for. Reused lemmas would be similar to both. Instead, we instead find a sharp decrease in similarity.

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, Llama3.1-8B-Instsr, and Qwen3-14B System Prompt

In the prompt (see Fig. 15) 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, Llama3.1-8B-Instsr, and Qwen3-14B displayed no obviously incorrect be-

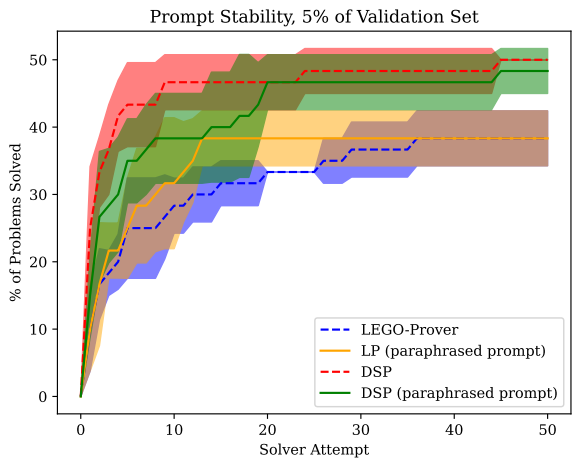

Figure 13: For LEGO-Prover and Draft-Sketch-Prove we compare 5 runs with our default prompt versus 5 runs with different paraphrased prompts. We evaluate on a subset of the validation set. We find that the systems are fairly robust to minor changes in wording – performance is typically within 1 standard deviation. These results are for a 5% subset of the validation split.

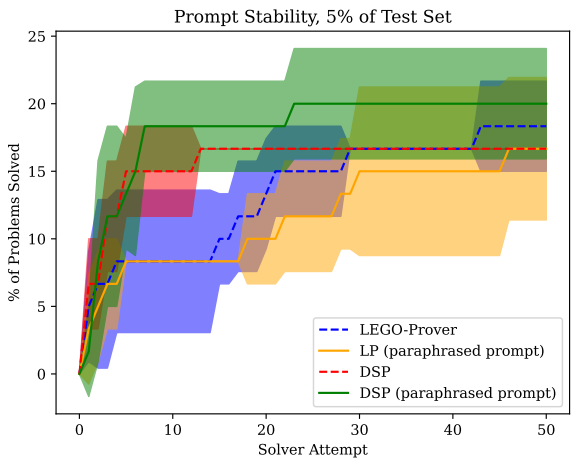

Figure 14: For LEGO-Prover and Draft-Sketch-Prove we compare 5 runs with our default prompt versus 5 runs with different paraphrased prompts. We evaluate on a subset of the test set. We find that these system are fairly robust to minor changes in wording – performance is typically within 1 standard deviation. These results are for a 5% subset of the test split.

haviours, 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 Fig. 16) 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 Fig. 17).

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 Fig. 18).

The EVOLVER prompts were unchanged. This prompt was also used for our Llama3.1 experiments.

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 Fig. 19).

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-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 Fig. 20).

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 15: 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 16: 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 17: 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 18: 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 19: LEGO-Prover: PROVER decomposer prompt for gpt-4o.

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 20: LEGO-Prover: PROVER decomposer prompt for o3-mini.

The PROVER's formalizer prompt was modified to re-enable Markdown formatting and to emphasize the desired output format (see Fig. 21).

The evolver prompts were likewise modified (see Fig. 22 through Fig. 26). The do_request prompt (see Fig. 22) was particularly problematic as o3-mini tended to disregard the formatting instructions.

D.2.4 Qwen3 prompts

Similar to o3-mini, we found it necessary to add formatting instructions to all EVOLVER prompts, in this case emphasizing that the output must be placed inside of a Markdown code block. The PROVER prompt had to be adjusted to further emphasize that the input problem should not be paraphrased. For brevity, and given the similarity to those used in o3-mini, we omit these six prompts – see `lego_prover/prompts/Experiment3/qwen3` in of the supplemental code for details.

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 Tab. 3 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, Fig. 7 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 Tab. 3 and plot Fig. 7 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

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 21: 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 22: LEGO-Prover: EVOLVER do_request 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 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 23: 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 24: 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 25: 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 26: LEGO-Prover: EVOLVER “scale complexity” prompt for o3-mini.

(`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 Fig. 27. Some lines have been word wrapped to fit within margins.

The retrieved lemma in Fig. 28 provided to the PROVER when generating the solution in Fig. 27 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 29 and 30 provided to the PROVER when generating the solution in Fig. 27 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 Fig. 31 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. It is reproduced almost exactly in the PROVER's output, with minor changes to parenthesization and the name of the lemma.

```
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 27: 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 28: Example lemma retrieved in the generation of the code in Fig. 27. 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 29: Example lemma retrieved in the generation of the code in Fig. 27. 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 30: Example lemma retrieved in the generation of the code in Fig. 27. 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 31: Example lemma retrieved in the generation of the code in Fig. 27. 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.

G TroVE and Agent Optimizer Experiments

G.1 TroVE Direct Reuse

TroVE (Wang et al., 2024e) 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.

When it comes to reuse we therefore evaluate how often the LLM uses existing helper functions from the library when it is prompted in the IMPORT mode. We outline direct reuse in TroVE under `CodeLlama-7B-Instruct` in Tables 11 through 18 and under `Llama3.1-8B-Instruct` in Tables 19 through 28, summing up function reuses over three seeds. However, as programs created in the IMPORT mode may not import a helper function at all, we report the number of programs that perform reuse at least once *and* are correct. We furthermore calculate the average length of a helper function by calculating the depth of its respective abstract syntax tree, as a proxy for how complex a function is.

As a result, among some domains such as Prealgebra and TabMWP we find strong evidence of reuse whereas on others such as Algebra and Precalculus no reuse is performed at all. Furthermore, the function lengths are rather short, with the longest program `median_of_numbers` calculating the median of a numeric column in a pandas DataFrame, which has a length of 11 (see Figure 32).

G.1.1 Trove `CodeLlama-7B-Instruct` Direct Reuse

```
def median_of_numbers(df, value_column):
    values = df[value_column].values
    values.sort()
    if len(values) % 2 == 0:
        median = (values[int(len(values) / 2)]
        + values[int(len(values) / 2) - 1]) / 2
    else:
        median = values[int(len(values) / 2)]
    return median

def num_divisors_common(n1, n2):
    return len(set([i for i in \
    range(1, min(n1, n2) + 1) \
    if n1 % i == 0 and n2 % i == 0]))

def area_triangle(a, b, c):
    return 0.5 * abs(a[0] * (b[1] - c[1]) \
    + b[0] * (c[1] - a[1]) \
    + c[0] * (a[1] - b[1]))
```

Figure 32: The three longest functions from TroVE's final libraries regarding their abstract syntax tree depth.

Algebra	
Correct Import Attempts:	1590/13215
Function Length:	–
No created function was reused.	

Table 11: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Algebra. Reuse aggregated over 3 runs.

Counting	
Correct Import Attempts:	720/4365
Function Length:	6.00 ± 0.82
`is_prime`	5
`is_perfect_square`	3
`has_unit_digit_3`	2

Table 12: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Counting.

Geometry	
Correct Import Attempts:	62/3555
Function Length:	–
No created function was reused.	

Table 13: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Geometry. Reuse aggregated over 3 runs.

Intermediate	
Correct Import Attempts:	101/7.545
Function Length:	–
No created function was reused.	

Table 14: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Intermediate. Reuse aggregated over 3 runs.

Number	
Correct Import Attempts:	895/7455
Function Length:	5.73 ± 1.29
`gcd (seed2)`	44
`gcd (seed3)`	33
`divisors`	31
`n_to_base_n`	13
`number_of_groups`	11
`number_of_soldiers_without_group`	11
`base_6`	10
`number_of_soldiers_without_group_estimate`	9
`most_likely_number_of_soldiers`	7
`most_likely_number_of_soldiers_estimate`	7
`lcm`	4
`find_digit_3`	1
`get_prime_factors`	1
`get_remainder`	1

Table 15: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Number. Reuse aggregated over 3 runs.

Prealgebra	
Correct Import Attempts:	1455/9540
Function Length:	6.25 ± 1.30
`count`	64
`gcd`	37
`is_prime`	16
`divisor`	9
`is_cube`	1
`is_sum_of_digits_equal_to_15`	1
`count_integers_with_sum_of_digits_equal_to_15`	0
`sum_k`	0

Table 16: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Prealgebra. Reuse aggregated over 3 runs.

Precalculus	
Correct Import Attempts:	87/2340
Function Length:	–
No created function was reused.	

Table 17: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for MATH Precalculus. Reuse aggregated over 3 runs.

TabMWP	
Correct Import Attempts:	7546/80640
Function Length:	6.04 ± 0.87
`total_cost (seed2)`	2716
`median_of_column`	1197
`total_cost`	1148
`sum_of_values`	652
`mode (seed3)`	625
`sum_column`	440
`mode`	383
`total_value`	377
`sum_of_values (seed3)`	349
`total_cost (seed3)`	277
`difference_between_scores`	273
`sum_of_values (seed2)`	73
`available_funds`	63
`range_of_numbers`	55
`median`	45
`max_value_per_column`	24
`score_difference_between`	21
`difference_in_values`	11
`range_of_values (seed3)`	11
`calculate_total_cost`	10
`largest_number_of_elements`	10
`difference_in_medals`	9
`get_available_funds`	8
`largest_number_of_pieces_of_pepperoni`	8
`range_of_values`	5

Table 18: Direct function reuse of TroVE under `CodeLlama-7B-Instruct` for TABMWP. Reuse aggregated over 3 runs.

G.1.2 Trove `Llama3.1-8B-Instruct` Direct Reuse

Algebra	
Correct Import Attempts:	957/13215
Function Length:	–
No created function was reused.	

Table 19: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Algebra. Reuse aggregated over 3 runs.

Counting	
Correct Import Attempts:	502/4365
Function Length:	7.00 ± 0.71
`count`	49
`check_palindrome`	8
`count_between`	7

Table 20: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Counting. Reuse aggregated over 3 runs.

Geometry	
Correct Import Attempts:	122/3555
Function Length:	–
No created function was reused.	

Table 21: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Geometry. Reuse aggregated over 3 runs.

Intermediate	
Correct Import Attempts:	194/7545
Function Length:	–
No created function was reused.	

Table 22: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Intermediate. Reuse aggregated over 3 runs.

Number	
Correct Import Attempts:	1037/7455
Function Length:	7.00 ± 1.73
`gcd (seed2)`	73
`num_divisors`	45
`is_prime`	37
`sum_of_consecutive_numbers`	33
`factorial`	31
`lcm (seed2)`	24
`num_divisors_common`	23
`sum_of_factors`	19

Table 23: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Number. Reuse aggregated over 3 runs.

Prealgebra	
Correct Import Attempts:	772/9540
Function Length:	5.92 ± 0.95
`num_of_numbers`	12
`diagonal_count`	6
`find_numbers`	6
`main`	6
`odd_integer`	6
`product`	6
`sum_of_digits`	6
`count_digits`	2
`smallest_multiple_of_6_greater_than`	2
`odd_digits`	1

Table 24: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Prealgebra. Reuse aggregated over 3 runs.

Precalculus	
Correct Import Attempts:	250/2340
Function Length:	10.00 ± 0.00
No created function was reused.	

Table 25: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for MATH Precalculus. Reuse aggregated over 3 runs.

TabMWP	
Correct Import Attempts:	16841/80640
Function Length:	6.50 ± 1.45
sum_column	2326
rate_of_change	2230
mode	1447
median	1442
mean_value	1381
median_value	1364
mode_of_values	1309
price_per_unit	1245
range_of_values	1220
cost_function	1200
mode_of_numbers	1192
diff_in_values	925
mean_value (seed3)	863
mean_of_numbers	754
get_row	515
get_value	503
total_price	349
max_value	293
min_value	260
get_value_at	165
total	126
cost	97
max_value (seed2)	88
min_frequency	71
get_range	17
median_of_numbers	12
score_difference	6
get_price	3
get_total	2
get_total_cost	2

Table 26: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for TABMWP. Reuse aggregated over 3 runs.

WTQ	
Correct Import Attempts:	4852/65160
Function Length:	6.71 ± 1.02
get_num_listings	938
get_value_by_condition	911
get_value_by_condition (seed3)	858
len_df_by_condition	836
get_most_common	384
get_value_by_condition (seed2)	318
get_total_score	290
get_first_value_by_condition	230
get_last_value (seed3)	221
get_first_value	214
get_next_match	169
get_last_value	154
get_earliest_date	99
get_position_by_conditions	78
get_min_value_by_condition	76
get_year_without_condition	64
get_consecutive_records	63
get_num_games_after_date	44
get_next_value	31
get_num_games_after	31
get_next_date	18
get_last_row	16
get_last_value (seed2)	15
get_total_points	11

Table 27: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for WTQ. Reuse aggregated over three runs.

HiTab	
Correct Import Attempts:	700/23610
Function Length:	5.00 ± 0.00
get_data_cell (seed2)	350
get_data_cell (seed3)	317
get_data_cell	206
get_row	95
get_column	65
get_ratio	58
get_data_cell_ratio	32

Table 28: Direct function reuse of TroVE under `Llama 3.1-8B-Instruct` for HITAB. Reuse aggregated over three runs.

G.2 Compute-Matched TroVE Evaluation

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.

Additionally, we discovered a weakness of TroVE's selection mechanism in that it performed majority voting two times - first on each of the 5 answers per mode, and second on the remaining 3 candidates. As this consistently decreases performance in comparison with direct majority voting on all 15 answers, we adapted the implementation at this point and report the results in Table 5 for CodeLlama-7B-Instruct and Llama3.1-8B-Instruct.

As depicted in Table 29 for CodeLlama, our ablation achieves results similar to TroVE, as well as for the reproduced variant as for the original reported TroVE results, which report the maximum value of 5 runs rather than the average. More strikingly, when using Llama3.1 the ablation consistently outperforms or matches TroVE across all domains (see Table 5). This observation is consistent with findings by Berlot-Attwell et al. (2024) who found that a CREATE-SKIP ensemble had comparable performance on MATH (Hendrycks et al., 2021b) – although 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).

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

G.3 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 31 through 35). 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 direct reuse behaviour (see Tab. 36). From manual inspection, the learned tools are often

	Ablation: Direct		TroVE	
Category	Original	Compute-Matched	Original	Reproduced
Algebra	15%	**26.7±0.5%**	25%	**29.1±0.7%**
Counting	14%	**23.6±0.4%**	26%	**26.9±2.2%**
Geometry	6%	**8.4±1.2%**	**8%**	7.7±1.1%
Intermediate	5%	**13.6±0.9%**	11%	**12.9±1.0%**
Number	16%	**28.0±1.8%**	25%	**29.6±0.9%**
Prealgebra	21%	**32.8±1.8%**	29%	**32.3±2.1%**
Precalculus	10%	**15.4±0.4%**	17%	**20.4±1.5%**
TabMWP	43%	**45.4±0.4%**	**47%**	40.1±2.6%

Table 29: Accuracy comparison between the compute-matched direct ablation and TroVE across different domains for CodeLlama 7B-Instruct. We report mean and standard deviation over 5 random seeds. Bold numbers mean that within the approach the setup performed best, underlined means that across both approaches the setup was best.

[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

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

Table 30: 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.

(thought not always) 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.

H Notable LEGO-Prover Lemmas

In this section we reproduce the only LEGO-Prover lemma to be reused verbatim. The lemma in Fig. 33 is indeed useful and broadly applicable. This successful attempt at library learning stands in contrast

```
lemma exponent_properties:
  fixes a :: real
  shows "a powr (m * n) = (a powr m) powr n"
  by (smt (z3) powr_powr)

lemma exponent_simplification:
  fixes a :: real
  shows "(a powr m) powr n = a powr (m * n)"
  by (metis exponent_properties)
```

Figure 33: This is the only lemma reused verbatim in our experiments. This is indeed a successful example of library learning, establishing a useful and general result: $a^{mn} = (a^m)^n$

```
fun f :: "real \<Rightarrow> real" where
  "f x = 5 * x + 4"

lemma f_x:
  fixes x :: real
  shows "5 * x + 4 = f x"
proof -
  have "f x = 5 * x + 4" by auto
  then show ?thesis by simp
qed
```

Figure 34: This is the only lemmas whose name is reused more than once. This lemma that $5x+4 = 5x+4$ is both trivial and overly specific; it is a failed example of library learning.

to the only lemma to name reused more than once (see Fig. 34).The lemma in Fig. 34 is trivial and specific to the equation $5x + 4$, making it a poor tool in the library.

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., 2024e) (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., 2024b)

	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 31: 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 32: 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 33: 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 34: 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 35: AgentOptimizer performance on MATH number theory. We report mean and standard deviation over 3 runs.

(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. (2024b) 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 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. (2024e) 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. (2024e) (https://github.com/zorazrw/trove/tree/c4d16b6a2e38020540db2a611fdea722da6b880c/data/tableqa).

All datasets are English datasets of mathematical

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 36: 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.

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.