# A Compute-Matched Re-Evaluation of TROVE on MATH

**Tobias Sesterhenn** [1] **Ian Berlot-Attwell** [2] **Janis Zenkner** [1] **Christian Bartelt** [1]

## Abstract

Reusing established theorems and formulas is central to mathematical problem solving, serving as essential building blocks for tackling increasingly complex challenges. Recent work, TROVE, argues that code-generating Large Language Models (LLMs) can benefit similarly on the MATH benchmark by inducing and reusing higher-level toolboxes. By allocating computational budget across an ensemble of three modes – directly generating code, creating tools, and reusing tools – TROVE claims to outperform a PRIMITIVE baseline that only performs direct generation. However, recent analysis (Berlot-Attwell et al., 2024) casts doubt on these gains, noting that the tools created are often trivial or rarely reused, suggesting that improvements may stem from self-consistency or self-correction. In this work, we re-evaluate TROVE on MATH, analyze the impact of each of its modes, and show that its benefit does not come from these mechanisms, but simply from a higher computational budget spent for TROVE compared to PRIMITIVE. To this end, we also perform a small correction in the original implementation of TROVE's selection mechanism, boosting TROVE's performance on MATH by $3\%$ in accuracy. After matching for compute, the benefit of TROVE reduces to a marginal improvement of $1\%$, suggesting that this toolbox approach does not provide a significant benefit on MATH.

## 1. Introduction

Reusing proven lemmas and formulas is how mathematicians keep cognitive load manageable and proofs short: recalling a schema (e.g., the Binomial theorem) frees working memory that would otherwise be spent on re-derivation (Sweller, 1988; Zhao et al., 2024), and automated–reasoning systems that reuse earlier proofs solve new theorems markedly faster than systems that start from scratch (Walther & Kolbe, 2000). In other words, good mathematics relies on a library of lemmas that can be imported rather than rebuilt.

This reuse principle has long guided library learning in computer science. Systems such as DreamCoder (Ellis et al., 2023) interleave solving tasks with compressing recurring code snippets into a growing library; the learned abstractions then make subsequent synthesis significantly faster and more sample-efficient. Later variants (Bowers et al., 2023; Cao et al., 2023; Grand et al., 2023) refine the idea, but the core insight remains: discover reusable components, store them, and apply them later. Recent work has explored how to leverage In-Context Learning (ICL), where a LLM is prompted with a description of the domain, examples, a set of available functions, and the task to solve. This allows for an incrementally extended library, or "toolbox", through a mechanism to select valid abstractions without retraining the model. These approaches (Wang et al., 2025; 2024b; Cai et al., 2023) have recently gained attention for their effectiveness in improving performance on various benchmarks.

Among them, TROVE (Wang et al., 2024b) currently stands out as a state-of-the-art method on the MATH dataset (Hendrycks et al., 2021). For a fixed computational budget of $K$ LLM calls, TROVE repeatedly generates Python code to solve a given task using three different prompting modes and then selects the most likely correct solution via self-consistency (i.e., majority vote) (Wang et al., 2022). The three modes are:

- **CREATE:** Solve the task and add any new functions to a toolbox that is periodically pruned.
- **IMPORT:** Solve the task by using functions already present in the toolbox.
- **SKIP:** Solve the task only with primitive, built-in functions, without using the toolbox.

Hence, when evaluating the use of CODELLAMA (Roziere et al., 2023), Wang et al. (2024b) reported that TROVE outperforms a PRIMITIVE baseline that operates only in SKIP

[1]Clausthal University of Technology, Clausthal, Germany [2]Vector Institute, University of Toronto, Toronto, Canada. Correspondence to: Tobias Sesterhenn <tobias.sesterhenn@tu-clausthal.de>.

*The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.* 

Table 1: Accuracy comparison across MATH categories. We report mean and standard variation on 5 different random seeds. The left two columns show the improvement in performance when matching PRIMITIVE for compute. On the right we reproduce the results of TROVE and show the performance gain when changing TROVE's selection mechanism from a two-stage to a one-stage approach. Green highlights represent strictly higher improvements for the compute-matched version of PRIMITIVE and the corrected version of TROVE.

| | **PRIMITIVE: Reproduction** | | **TroVE: Reproduction** | | |
|---|---|---|---|---|---|
| Category | Original Results | Compute-Matched | Original Results | Reproduced | Improved |
| Algebra | 0.15 | **0.27 ± 0.01** | 0.25 | 0.26 ± 0.01 | **0.29 ± 0.01** |
| Counting | 0.14 | **0.24 ± 0.00** | 0.26 | 0.24 ± 0.02 | **0.27 ± 0.02** |
| Geometry | 0.06 | **0.08 ± 0.01** | 0.08 | 0.05 ± 0.01 | **0.08 ± 0.01** |
| Intermediate | 0.05 | **0.14 ± 0.01** | 0.11 | 0.12 ± 0.02 | **0.13 ± 0.01** |
| Number | 0.16 | **0.28 ± 0.02** | 0.25 | 0.27 ± 0.01 | **0.30 ± 0.01** |
| Prealgebra | 0.21 | **0.33 ± 0.02** | 0.29 | 0.29 ± 0.02 | **0.32 ± 0.02** |
| Precalculus | 0.10 | **0.15 ± 0.01** | 0.17 | 0.18 ± 0.03 | **0.20 ± 0.01** |
| MATH | 0.12 | **0.24 ± 0.01** | 0.20 | 0.22 ± 0.01 | **0.25 ± 0.01** |

mode achieving 20% on MATH compared to PRIMITIVE only solving 12%.

However, Berlot-Attwell et al. (2024) raise the question whether the TROVE mechanism actually increases performance as they show that tools are either trivial (already contained in the knowledge of the LLM) or never reused (see Appendix B): An ablation study in a subset of MATH shows that the removal of the IMPORT mode does not negatively affect performance. Although the authors hypothesize TROVE's benefit may come from self-consistency or the self-correctness mechanism, the question of what makes TROVE better remains an open question.

In this work, we investigate what really drives TROVE's strong performance on the MATH benchmark and show that its main advantage comes from a higher computational budget rather than the toolbox mechanism itself. We first test whether TROVE continues to outperform a PRIMITIVE baseline when both systems are given the same number of LLM calls. To ensure a fair comparison, we also fix a discrepancy between TROVE's described agreement-based selection method and its actual implementation, correcting an error that improves its performance by 3% accuracy. Finally, we analyze whether TROVE's varied prompting strategies provide benefits independently of the toolbox, and how TROVE and PRIMITIVE scale with increased compute under an ideal selection mechanism.

Our results show that while TROVE marginally outperforms PRIMITIVE, the accuracy gain reduces to only 1% when controlling for the computational budget. We also observe that, under compute-matched settings, TROVE produces a slightly more diverse set of candidate predictions — on average 0.74 more per task. While this may offer some benefit by broadening the search space, it also increases the difficulty in selecting the correct answer. Our findings indicate that TROVE's improved performance is best explained by a higher allocation of the computational budget, rather than by its toolbox mechanism.

## 2. Revisiting TROVE

**Prompting Modes** TROVE is a training-free method to build a toolbox of reusable high-level functions to solve programmatic tasks, operating without ground truth labels or external supervision (Wang et al., 2024b). For each task, TROVE samples $K$ candidate programs using a fixed computational budget, divided equally among three distinct prompting modes: SKIP, CREATE, and IMPORT. In SKIP mode, the model generates a solution using only primitive Python functions that, for MATH, are inherently stored in the LLM weights, i.e., parametric knowledge. For example, the LLM can call *numpy* or *sympy* functions. In CREATE mode, the model is instructed to define a new helper function that encapsulates reusable logic and adds it to the toolbox. This function is designed to support solving the current task and potentially benefit future tasks. In IMPORT mode, the model can select and apply existing functions from the current toolbox to solve the task. The original work compares TROVE against a PRIMITIVE baseline. Hereby, the baseline implements the same prompting mode as SKIP. Thus, when fairly comparing PRIMITIVE and TROVE, each TROVE mode is used $\frac{K}{3}$ times, whereas PRIMITIVE is used $K$ times.

**Agreement-Based Selection** To decide on a final candidate, the authors propose an agreement-based selection algorithm using majority voting and solution complexity: among the candidates that execute successfully (i.e., without errors), the most frequent answer is chosen. If there is a tie, the solution implemented by the shortest program, in terms of operations, is selected (see Algorithm 2).

## 3. Experimental Setup

**Compute-Matching** In this work, we analyze whether TROVE's methodology using library learning and three different prompting styles brings an advantage over a PRIMITIVE baseline. For this we reproduce the results of the original work by Wang et al. (2024b) and match PRIMITIVE for a computational budget of $K = 15$ LLM calls. In line with the original work, we evaluate the approaches using CODELLAMA2-7B-INSTRUCT (Roziere et al., 2023). We choose the hyperparameters accordingly, as described in Appendix A. All experiments are run five times, and we report the mean and standard variation of the accuracies in each category of MATH. Here, a task is correctly solved if the selection mechanism selects a candidate whose execution result agrees with the task solution.

**Oracle Selection Mechanism** In the experiments, we also focus on the potential of TROVE under an *oracle selection mechanism*, which assumes a perfect selector that identifies the correct answer as soon as it appears in the candidate set. We report the corresponding performance using *pass@K* for a budget of $K$.

**MATH Dataset** The MATH dataset introduced by Hendrycks et al. (2021) consists of $12,500$ competition-style math problems. In this work, we use a subset of $3,201$ tasks, following Wang et al. (2024b). The dataset is divided into seven categories: 881 tasks in *prealgebra*, 291 tasks in *algebra*, 237 in *number theory*, 503 in *counting & probability*, 497 in *geometry*, 636 in *intermediate algebra*, and 156 tasks in *precalculus*. Each task consists of a query in natural language and a numeric ground-truth answer.

## 4. Analysis

We first reproduce the experiments of the original work, but matching the computational budget (§4.1.1) and correcting a discrepancy in the agreement-based selection mechanism (§4.1.2). We further investigate the impact of the different modes on the overall performance of TROVE (§4.2) and analyze the performance of PRIMITIVE and TROVE under different computational budgets (§4.3).

### 4.1. Correcting Original TROVE Results

#### 4.1.1. MATCHING COMPUTATIONAL BUDGET

To strengthen the hypothesis that TROVE's toolbox mechanism has no significant impact on performance, we compare TROVE against the compute-matched PRIMITIVE baseline across all MATH domains by sampling the LLM with $K = 15$ times per task. We further reproduce the reported results of the TROVE approach, achieving $\pm 3\%$ of the original reported results over five random seeds.

As depicted in Table 1, the compute-matched PRIMITIVE baseline now, instead of reported $12\%$, achieves $24\%$ in MATH, outperforming the original $22\%$ reported by TROVE. Among the different categories, the performance differs at most $\pm 4\%$ in accuracy between PRIMITIVE and TROVE with PRIMITIVE performing better in algebra, geometry, intermediate algebra, number and prealgebra. When analyzing the performance across different computational budgets, the results suggest that the reported results of PRIMITIVE in the original work were achieved by setting $K$ to 1 instead of 15 (see Appendix B.1, Table 5). Thus, contrary to the claim of the authors, TROVE appears to benefit primarily from repeated sampling rather than toolbox learning.

#### 4.1.2. CORRECTING TROVE'S SELECTION MECHANISM

**Algorithm 1** Two-Stage Candidate Selection

**function** MULTIWAYGENERATION(example, K)
  $candidates \leftarrow []$
  **for all** mode **in** {IMPORT, CREATE, SKIP} **do**
    $C \leftarrow$ SAMPLEMODEL$(example, mode, K)$
    $i^* \leftarrow$ SELECTBEST$(C)$
    APPEND$(candidates,\ C[i^*])$
  $j^* \leftarrow$ SELECTBEST$(bestResp)$
  **return** $candidates[j^*]$.mode, $candidates[j^*]$

**Algorithm 2** One-Stage Candidate Selection

**function** MULTIWAYGENERATION(example, K)
  $candidates \leftarrow [\,]$
  **for all** mode **in** {IMPORT, CREATE, SKIP} **do**
    $c \leftarrow$ SAMPLEMODEL(example, mode, $K$)
    EXTEND(candidates, c)
  $i^* \leftarrow$ SELECTBEST$(candidates)$
  **return** $candidates[i^*]$.mode, $candidates[i^*]$

When reviewing TROVE, we also identified a difference between the algorithm described in the original work and its implementation in Python[1]. The original implementation proposes a single stage agreement-based selection mechanism, based on self-consistency and solution complexity on the $K$ candidate responses (Wang et al., 2024b). However, the implementation performs a two-stage candidate selection, where in the first stage for each of the three modes, one of $\frac{K}{3}$ candidates is chosen individually and then the algorithm is applied to the three remaining candidates (see Algorithm 1).

However, this leads to a lower probability of selecting a correct answer, as the candidate is chosen by majority voting on a much smaller size. Once we correct the implementation for a one-stage candidate selection, as described in Algorithm 2, TROVE's performance is consistently improved

[1] https://github.com/zorazrw/trove/blob/main/run_trove.py

across all categories. Table 1 shows that compared to our reproduced results, the performance increases by 3% on MATH. Compared to the originally reported results, it increases by 5%. Through this, TROVE slightly outperforms PRIMITIVE by 1%, although these results are not significant (we show our findings with 5 random seeds).

### 4.2. Effects of TROVE's diverse prompting

After matching the compute, PRIMITIVE performs almost equally to TROVE with TROVE performing only slightly better after changing to the one-stage selection mechanism. However, TROVE's three-mode prompting mechanism brings more diversity into the approach, as the PRIMITIVE prompting mode only matches the SKIP mode. Therefore, in the following, we further analyze the impact of each prompt mode on the overall performance and compare TROVE's candidate proposals with the ones by PRIMITIVE.

To examine the potential of both PRIMITIVE and TROVE, in the following we use the oracle selection mechanism for candidate selection and report *pass@K*.

#### 4.2.1. PERFORMANCE OF PROMPT MODES

Figure 1 shows the distribution of the tasks solved in the different modes for $pass@5$ and across 5 seeds. Thereby, the CREATE mode performs best across all categories, followed by SKIP and IMPORT. However, TROVE profits from all three modes, as they partially solve different tasks on the MATH dataset (Table 2). While CREATE uniquely solves the highest proportion of tasks, namely 8% in the entire dataset, SKIP and IMPORT uniquely solve 6% and 4%, respectively.

Furthermore, in Appendix B.3 we show that TROVE also solves tasks that are not solved by PRIMITIVE (Table 6) and that the solutions found by PRIMITIVE have the highest overlap with those found by the SKIP mode of TROVE (Figure 3). This is reasonable since both PRIMITIVE and SKIP use the same prompt. In accordance with this, PRIMITIVE is least similar to IMPORT in their sets of solved challenges, since their prompting styles differ the most.

#### 4.2.2. PROMPT DIVERSITY INCREASES HYPOTHESIS SPACE PER TASK

Furthermore, prompt diversity actually leads to a higher variety of proposed solutions per task, suggesting that TROVE is capable of covering a larger hypothesis space. We quantify this by counting the mean number of different predictions produced per task in the fixed compute budget of $K = 15$ LLM calls. Table 3 shows that on counting, number, prealgebra, and precalculus, TROVE creates approximately one additional solution candidate, with, on average, 0.74 additional solutions on any MATH task. However, while the increased number of candidate solutions may be beneficial when combined with a good selection mechanism, this may not hold for a weaker one such as majority voting, where the extra predictions inject noise.

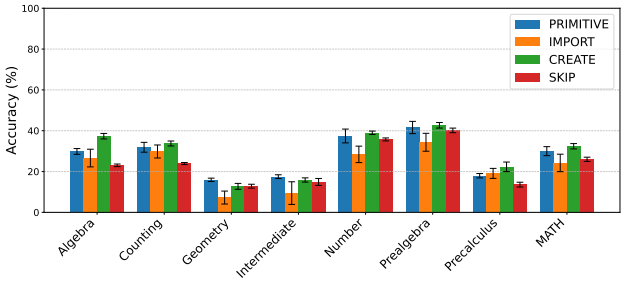


Figure 1: Distribution of solved tasks across the different categories for each mode. We report $pass@5$, i.e., using the oracle selection mechanism for a computational budget of 5.

Table 2: Fraction of tasks uniquely solved by each mode (mean ± standard variation over seeds). The MATH row aggregates across all categories.

| Category | IMPORT | CREATE | SKIP |
|---|---|---|---|
| Algebra | $0.04 \pm 0.01$ | $\mathbf{0.10 \pm 0.02}$ | $0.04 \pm 0.01$ |
| Counting | $0.05 \pm 0.01$ | $\mathbf{0.07 \pm 0.01}$ | $0.05 \pm 0.01$ |
| Geometry | $0.03 \pm 0.01$ | $0.07 \pm 0.02$ | $\mathbf{0.07 \pm 0.01}$ |
| Intermediate | $0.03 \pm 0.02$ | $0.07 \pm 0.02$ | $\mathbf{0.08 \pm 0.01}$ |
| Number | $0.05 \pm 0.01$ | $\mathbf{0.08 \pm 0.01}$ | $0.06 \pm 0.01$ |
| Prealgebra | $0.05 \pm 0.01$ | $\mathbf{0.07 \pm 0.01}$ | $0.07 \pm 0.00$ |
| Precalculus | $0.04 \pm 0.02$ | $\mathbf{0.06 \pm 0.02}$ | $0.03 \pm 0.01$ |
| MATH | $0.04 \pm 0.00$ | $\mathbf{0.08 \pm 0.01}$ | $0.06 \pm 0.00$ |

Table 3: Mean number and standard variation of distinct solutions per task under a fixed compute budget of 15 LLM calls. Positive Δ values indicate that TROVE proposes a higher variety of predictions compared to the PRIMITIVE baseline.

| Category | PRIMITIVE | TROVE | Δ |
|---|---|---|---|
| Algebra | $4.20 \pm 0.04$ | $4.83 \pm 0.15$ | +0.64 |
| Counting | $5.94 \pm 0.06$ | $6.77 \pm 0.13$ | +0.83 |
| Geometry | $6.79 \pm 0.66$ | $6.64 \pm 0.41$ | -0.15 |
| Intermediate | $3.89 \pm 0.06$ | $4.09 \pm 0.25$ | +0.20 |
| Number | $4.88 \pm 0.64$ | $5.78 \pm 0.19$ | +0.90 |
| Prealgebra | $5.18 \pm 0.04$ | $6.47 \pm 0.16$ | +1.29 |
| Precalculus | $3.36 \pm 0.12$ | $4.83 \pm 0.14$ | +1.47 |
| MATH | $4.76 \pm 0.19$ | $5.50 \pm 0.19$ | +0.74 |

### 4.3. Effect on different computational budgets

To analyze whether TROVE's diverse prompting leads to better pass@K results across growing computational budgets, we calculate the accuracies over different computational budgets. However, Figure 2 shows that there is no significant difference between TROVE and PRIMITIVE. Whereas TROVE solves 44% of tasks with a budget of $K = 15$, PRIMITIVE achieves 43% accuracy. Figure 4 in Appendix B.4 shows that this gap remains consistent for a budget of up to $K = 75$. Notably, the results also highlight the importance of the selection mechanism. When using an oracle selector both TROVE and PRIMITIVE achieve 19% higher accuracy compared to their respective majority-vote baselines. This suggests that further gains are more likely to come from better selection rather than increased prompt diversity alone.

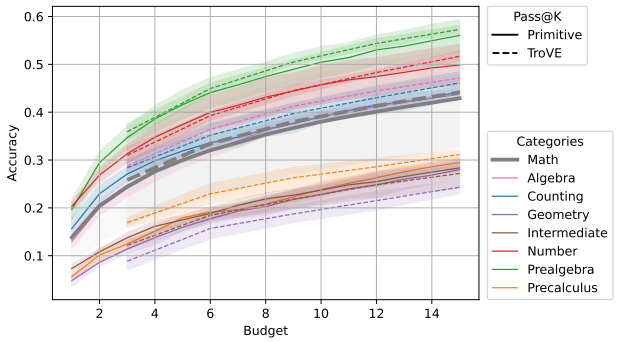


Figure 2: Performance across different computational budgets represented by the sampling sizes for TROVE and PRIMITIVE on the MATH dataset. As TROVE requires each mode to be called, the plot depicts the performance only for multiples of 3.

## 5. Related Work

### 5.1. Library Learning on Formal Syntaxes

Early work in library learning leverages domain-specific languages (DSLs) to constrain the search space and extract reusable abstractions. DreamCoder alternates between a neural–guided enumerative *wake* phase that finds programs in a functional DSL and a *sleep* phase that compresses recurring subprograms into a growing library of abstractions (Ellis et al., 2023). Based on this, STITCH employs a corpus-driven, top-down search to discover higher-level primitives without exhaustive enumeration (Bowers et al., 2023), while Babble uses e-graphs and anti-unification to merge and refine candidate subexpressions into a library (Cao et al., 2023). LILO further augments DreamCoder by integrating an LLM to propose promising enumerative candidates alongside classic search, improving both scalability and solution quality (Grand et al., 2023).

### 5.2. LLM Program Creation on MATH

More recent approaches drop the DSL constraint and operate directly in general-purpose languages via in-context "toolboxes." LATM prompts an LLM once per task to generate exactly one helper function from Big-Bench, then uses that toolbox to solve the task (Cai et al., 2023). CREATOR and CRAFT both adopt multi-iteration pipelines: CREATOR cycles through generate–refine–execute stages (Qian et al., 2023), while CRAFT adds abstraction, validation, deduplication, and retrieval steps to assemble a specialized toolset (Yuan et al., 2023). ReGAL, which was published parallely to TROVE, applies offline refactoring and pruning cycles to Python solutions on MATH (Stengel-Eskin et al., 2024). A recent study suggests that TROVE outperforms CREATOR, LATM, and CRAFT on the MATH benchmark in both accuracy and efficiency (Wang et al., 2024a).

Within automated theorem proving, several works extract reusable formal lemmas from ground-truth proofs. ATG (Lin et al., 2024) generates new, reusable theorems from the Metamath library to improve downstream proof search, while REFACTOR (Zhou et al., 2024) extracts modular lemmas from existing proofs to shorten and strengthen theorem proving. LEGO-Prover (Wang et al., 2023) introduces a dynamic library of verified lemmas and leverages an LLM to construct proofs modularly, enabling the system to tackle increasingly complex theorems more effectively.

### 5.3. Evaluation of Tool Use and Library Learning

Empirical analyses have begun to question whether toolbox learning truly drives gains over simple sampling. Most related to our work, Berlot-Attwell et al. (2024) demonstrate that for TROVE and LEGO-Prover most "learned" libraries are invoked only once, casting doubt on their reusability. In a concurrent study, they show that for LEGO-Prover the benefit of the toolbox mechanism vanishes, as soon as the computational cost is taken into account (Berlot-Attwell et al., 2025). Regarding sampling in Program Synthesis, Li & Ellis (2024) show that finetuning an LLM and repeatedly sampling from it significantly outperforms library learning methods such as LILO (Grand et al., 2023) and ReGAL (Stengel-Eskin et al., 2024). However, in contrast to TROVE these approaches leverage an offline dataset to create the library, whereas TROVE creates its toolbox online during inference. More broadly, Yue et al. (2025) find that increasing the number of samples from a base LLM often outperforms reinforcement learning methods under a fixed compute budget. This observation aligns with our findings on TROVE, suggesting that, in some cases, simply allocating more compute to sampling from a primitive model can match or exceed the performance of more complex mechanisms such as toolbox construction.

## 6. Conclusion

The findings of this study indicate that the primary advantage of the TROVE approach comes less from the use of an incrementally assembled toolbox of abstractions and more from the increased probability of finding the correct solutions through repeated sampling. When controlling for the total number of generated solutions, the baseline approach that simply samples more candidate programs using its inherent domain knowledge performs comparable to TROVE. In the MATH domain, we confirm that library learning does not bring a significant advantage over this simple baseline. However, our experiments show that TROVE's prompting mechanism does lead to an increased hypothesis space. Under a stronger selection mechanism, this may lead to a small benefit over PRIMITIVE, which exists in our experiments but under majority selection we demonstrate it is not significant. More importantly, a driving factor is the choice of the selection mechanism. Although TROVE and PRIMITIVE often create a correct candidate, the selection mechanism often does not select it, as shown by the $19\%$ difference between oracle selection and majority voting for $K = 15$. Improving the current selection mechanism would lead to a much higher benefit than the one currently brought from TROVE over the baseline.

In conclusion, while our results cast doubt on whether tool-based methods represent an alternative to intensive sampling in MATH, we remain optimistic about the long-term potential of systematic abstraction learning for LLMs in other domains. Recent work on agentic tasks (Wang et al., 2025; Zheng et al., 2025) has shown that tools can effectively compress past experiences that can then be reused to improve efficiency and solving harder long horizon challenges. By deepening understanding on when and how library learning can assist LLMs, future research can more effectively utilize toolboxes to achieve improvements in complex problem solving.

## 7. Impact Statement

This paper advances Library Learning by analyzing the effectiveness of toolbox-based approaches for program generation with LLMs in mathematical problem solving. Our findings suggest that reported improvements from toolbox reuse may largely stem from increased sampling rather than true abstraction learning. This informs a broader understanding of when abstraction mechanisms in LLMs are beneficial, potentially guiding more efficient prompting and evaluation strategies. From an ethical perspective, this work does not involve sensitive data, user interaction, or deployment in high-stakes environments. While insights may influence future tool design for educational or productivity applications, we foresee no immediate societal risks.

## References


Berlot-Attwell, I., Rudzicz, F., and Si, X. Library learning doesn't: The curious case of the single-use" library". *arXiv preprint arXiv:2410.20274*, 2024.

Berlot-Attwell, I., Rudzicz, F., and Si, X. Llm library learning fails: A lego-prover case study. *arXiv preprint arXiv:2504.03048*, 2025.

Bowers, M., Olausson, T. X., Wong, L., Grand, G., Tenenbaum, J. B., Ellis, K., and Solar-Lezama, A. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023.

Cai, T., Wang, X., Ma, T., Chen, X., and Zhou, D. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

Cao, D., Kunkel, R., Nandi, C., Willsey, M., Tatlock, Z., and Polikarpova, N. Babble: Learning better abstractions with e-graphs and anti-unification. *Proceedings of the ACM on Programming Languages*, 7(POPL):396–424, 2023.

Cheng, Z., Dong, H., Wang, Z., Jia, R., Guo, J., Gao, Y., Han, S., Lou, J.-G., and Zhang, D. Hitab: A hierarchical table dataset for question answering and natural language generation. *arXiv preprint arXiv:2108.06712*, 2021.

Ellis, K., Wong, L., Nye, M., Sable-Meyer, M., Cary, L., Anaya Pozo, L., Hewitt, L., Solar-Lezama, A., and Tenenbaum, J. B. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220050, 2023.

Grand, G., Wong, L., Bowers, M., Olausson, T. X., Liu, M., Tenenbaum, J. B., and Andreas, J. Lilo: Learning interpretable libraries by compressing and documenting code. *arXiv preprint arXiv:2310.19791*, 2023.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Li, W.-D. and Ellis, K. Is programming by example solved by llms? *arXiv preprint arXiv:2406.08316*, 2024.

Lin, X., Cao, Q., Huang, Y., Yang, Z., Liu, Z., Li, Z., and Liang, X. Atg: Benchmarking automated theorem generation for generative language models. *arXiv preprint arXiv:2405.06677*, 2024.

Lu, P., Qiu, L., Chang, K.-W., Wu, Y. N., Zhu, S.-C., Rajpurohit, T., Clark, P., and Kalyan, A. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022.

Pasupat, P. and Liang, P. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

Qian, C., Han, C., Fung, Y. R., Qin, Y., Liu, Z., and Ji, H. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. *arXiv preprint arXiv:2305.14318*, 2023.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Stengel-Eskin, E., Prasad, A., and Bansal, M. Regal: Refactoring programs to discover generalizable abstractions. *arXiv preprint arXiv:2401.16467*, 2024.

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

Walther, C. and Kolbe, T. Proving theorems by reuse. *Artificial Intelligence*, 116(1-2):17–66, 2000.

Wang, H., Xin, H., Zheng, C., Li, L., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., et al. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wang, Z., Cheng, Z., Zhu, H., Fried, D., and Neubig, G. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*, 2024a.

Wang, Z., Fried, D., and Neubig, G. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024b.

Wang, Z., Gandhi, A., Neubig, G., and Fried, D. Inducing programmatic skills for agentic tasks. *arXiv e-prints*, pp. arXiv–2504, 2025.

Yuan, L., Chen, Y., Wang, X., Fung, Y. R., Peng, H., and Ji, H. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428*, 2023.

Yue, Y., Chen, Z., Lu, R., Zhao, A., Wang, Z., Song, S., and Huang, G. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Zhao, B., Lucas, C. G., and Bramley, N. R. A model of conceptual bootstrapping in human cognition. *Nature Human Behaviour*, 8(1):125–136, 2024.

Zheng, B., Fatemi, M. Y., Jin, X., Wang, Z. Z., Gandhi, A., Song, Y., Gu, Y., Srinivasa, J., Liu, G., Neubig, G., et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Zhou, J. P., Wu, Y., Li, Q., and Grosse, R. Refactor: Learning to extract theorems from proofs. *arXiv preprint arXiv:2402.17032*, 2024.

## A. Experimental Setup

We used the hyperparameters specified in the original TROVE paper (Wang et al., 2024b), except for the PRIMITIVE baseline that was extended for different values of the num_return_sequences parameter, which defines the number of LLM calls per task. Furthermore, per MATH domain, the PRIMITIVE baseline and TROVE were run for five different seeds. A slight adaptation was performed by setting the *exec_timeout* value from $100s$ to $30s$, but this did not lead to any performance drop. We used the official implementation from Github, which is released under the CC-BY-SA-4.0 license.

The experiments were run on a single NVIDIA RTX A6000, 2 CPUs, and 48GB RAM. Our code and data are available at https://github.com/Tsesterh/TroVE_Compute_Matched.

Table 4: Hyperparameters for TroVE and the PRIMITIVE baseline.

| **Hyperparameter** | **TroVE** | **PRIMITIVE** |
|---|---|---|
| trim_steps | 500 | 500 |
| exec_timeout | 100 | 100 |
| top_p | 0.95 | 0.95 |
| num_return_sequences | 5 | 1, ..., 15 |
| temperature | 0.6 | 0.6 |
| max_new_tokens | 512 | 512 |
| suffix | — | primitive |

## B. Additional Experimental Results

In this section, we provide additional insights by analyzing TROVE and PRIMITIVE on different topics. In §B.1 we show that the results of the original work probably were produced by prompting PRIMITIVE only once. In §B.2 we analyze differences in the sets of solutions found by TROVE and PRIMITIVE, evaluating the question whether TROVE's diverse prompting mechanisms can solve challenges that cannot be solved by PRIMITIVE. In §B.3 we further analyze the similarity of the proposed solutions between PRIMITIVE and each of TROVE's modes. In §B.4 we analyze the potential of each mode for an increased computational budget, combining the results of different seeds to evaluate for $K = 75$. Lastly, in §B.5 we analyze whether there is a difference between the difficulty levels that both approaches solve on MATH.

### B.1. Reproducing the TROVE paper

To test the hypothesis that the PRIMITIVE baseline is outperformed by TROVE due to the computational budget, we reproduce the results for different computational budgets $K$. As can be seen in Table 5 for a computational budget of $K = 1$ the performance of the PRIMITIVE baseline is closest to the performance reported in the original paper. Furthermore, when aligned for compute, PRIMITIVE performs equally as TROVE.

Table 5: Performance of PRIMITIVE and TROVE for different computational budgets using the agreement-based selection mechanism (for TROVE we report the results for the improved version of the mechanism). We report for multiples of 3 as for TROVE each of the three modes needs to be called. $P@K$ stands for calling the PRIMITIVE mode $K$ times, $T@K$ for calling TROVE mode $K$ times. As for $K = 1$ no TROVE value exists, we add the originally reported value of PRIMITIVE, to show that it was indeed not aligned for compute, as the values are within $2\%$ on the whole MATH dataset. We report the mean values across 5 seeds.

| Category | P@1 | P (orig) | P@3 | T@3 | P@6 | T@6 | P@9 | T@9 | P@12 | T@12 | P@15 | T@15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| algebra | 0.13 | 0.15 | 0.19 | **0.23** | 0.23 | **0.26** | 0.25 | **0.28** | 0.26 | **0.29** | 0.27 | **0.29** |
| counting | 0.16 | 0.14 | 0.20 | **0.20** | 0.22 | **0.24** | 0.22 | **0.26** | 0.22 | **0.27** | 0.24 | **0.27** |
| geometry | 0.05 | 0.06 | **0.06** | 0.05 | **0.08** | 0.06 | **0.08** | 0.07 | **0.08** | 0.07 | **0.08** | 0.08 |
| intermediate | 0.07 | 0.05 | 0.10 | **0.10** | **0.12** | 0.12 | 0.12 | **0.13** | 0.13 | **0.13** | **0.14** | 0.13 |
| number | 0.20 | 0.16 | **0.24** | 0.24 | 0.26 | **0.26** | 0.28 | **0.28** | 0.28 | **0.29** | 0.28 | **0.30** |
| prealgebra | 0.20 | 0.21 | **0.26** | 0.25 | **0.30** | 0.29 | **0.32** | 0.31 | **0.32** | 0.32 | **0.33** | 0.32 |
| precalculus | 0.06 | 0.10 | 0.10 | **0.13** | 0.12 | **0.17** | 0.14 | **0.18** | 0.14 | **0.20** | 0.15 | **0.20** |
| MATH | 0.14 | 0.12 | 0.18 | **0.19** | 0.21 | **0.22** | 0.23 | **0.24** | 0.23 | **0.25** | 0.24 | **0.25** |

Table 6: Exclusive task coverage by TROVE and PRIMITIVE across MATH categories. For each method, we report the number and percentage of tasks solved (i) consistently across all five seeds but by no seed of the other method (consistent gain), and (ii) by at least one seed but not all seeds, and not by any seed of the other method (potential gain).

| | **TROVE** | | | | **PRIMITIVE** | | | |
|---|---|---|---|---|---|---|---|---|
| | Consistent gain | | Potential gain | | Consistent gain | | Potential gain | |
| Category | # | % | # | % | # | % | # | % |
| Algebra | 25 | 2.84% | 83 | 9.42% | 10 | 1.14% | 84 | 9.53% |
| Counting | 3 | 1.03% | 41 | 14.09% | 5 | 1.72% | 18 | 6.19% |
| Geometry | 0 | 0.00% | 18 | 7.59% | 1 | 0.42% | 26 | 10.97% |
| Intermediate | 2 | 0.40% | 60 | 11.93% | 7 | 1.39% | 32 | 6.36% |
| Number | 1 | 0.20% | 49 | 9.86% | 0 | 0.00% | 37 | 7.44% |
| Prealgebra | 2 | 0.31% | 64 | 10.06% | 2 | 0.31% | 60 | 9.43% |
| Precalculus | 1 | 0.64% | 18 | 11.54% | 0 | 0.00% | 12 | 7.69% |
| **MATH** | 33 | 1.03% | 265 | 8.28% | 24 | 0.75% | 218 | 6.81% |

### B.2. Coverage Diversity

To quantify the additional coverage the diversity of the three modes brings, we separate the additional benefit of TROVE over PRIMITIVE into two parts, assuming a perfect selection mechanism:

1. **Consistent gain** – tasks consistently solved by *every* TROVE seed and by *no* PRIMITIVE seed.
2. **Potential gain** – tasks solved by *at least one* TROVE seed, by *no* PRIMITIVE seed, but *not* solved by all TROVE seeds. These represent tasks in the hypothesis space that can be reached basically in addition by a TROVE seed but not by a PRIMITIVE seed.

Table 6 shows both the numbers per MATH category and for the benchmark as a whole, as well for TROVE and for PRIMITIVE. The results show that over the different TROVE seeds the method consistently solves $1.03\%$ of tasks that were not solved by any PRIMITIVE run. Potentially, on the five seeds, TROVE additionally solves $8.28\%$ of the tasks in any seed that were not found by any PRIMITIVE seed. On the other hand, PRIMITIVE also solves $0.75\%$ that were not solved by any TROVE run. Therefore, the results show that the different prompt modes do not necessarily help TROVE solve a larger variety of challenges.

### B.3. Comparing PRIMITIVE and TROVE solutions

For the cross–type analysis we compare every PRIMITIVE run with every run of a given TROVE mode $M$. Let

$$\mathcal{S}_{\mathrm{P}} = \{1, \ldots, k\}, \qquad \mathcal{S}_M = \{1, \ldots, \ell\}$$

be the sets of indexes of random seeds for PRIMITIVE and for mode $M$, and denote by $A_s^{\mathrm{P}}$ and $A_t^M$ the corresponding sets of solved challenges. The *cross–type mean Jaccard similarity* between PRIMITIVE and the mode $M$ is then

$$\overline{J}_{\mathrm{P}\leftrightarrow M} \;=\; \frac{1}{k\,\ell} \sum_{s\in\mathcal{S}_{\mathrm{P}}} \sum_{t\in\mathcal{S}_M} \underbrace{\frac{|A_s^{\mathrm{P}} \cap A_t^M|}{|A_s^{\mathrm{P}} \cup A_t^M|}}_{J(A_s^{\mathrm{P}}, A_t^M)},$$

i.e., the average Jaccard similarity over all $k \times \ell$ pairs of runs from the two types. We compute this value separately for each TROVE mode (SKIP, IMPORT, CREATE) and for every problem category.

As depicted in Figure 3, when computing the Jaccard similarity between PRIMITIVE and the three modes, SKIP and PRIMITIVE have the highest value (Figure 3). This makes sense as SKIP and PRIMITIVE share the same prompt. Furthermore, PRIMITIVE is least similar to IMPORT, as their prompting styles differ the most. However, this does not count for each

category, as for precalculus the similarity to IMPORT is actually highest. We suggest that the reason for this is that IMPORT outperforms SKIP in precalculus, as shown in Figure 1, resulting in a potentially greater overlap with PRIMITIVE.

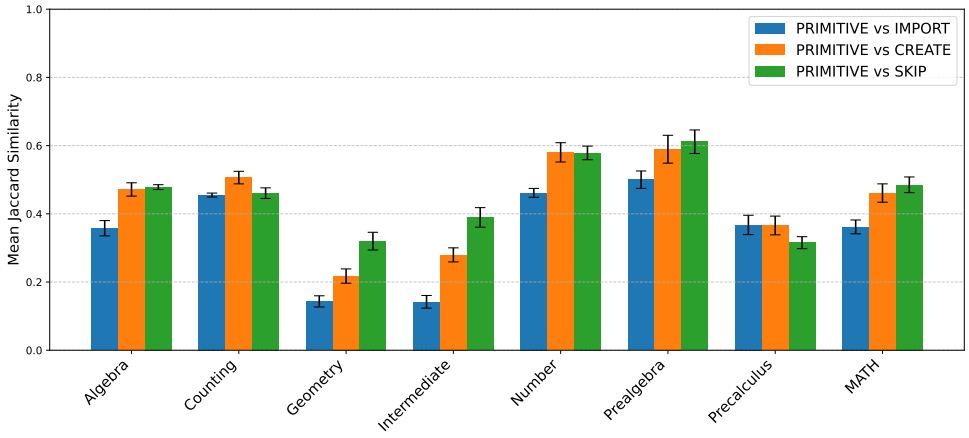


Figure 3: Mean Jaccard Similarities over 5 random seeds between PRIMITIVE and each TROVE mode for solutions found by the oracle selection mechanism.

### B.4. Impact of Selection Mechanism

We analyze the potential of each approach for an increasing computational budget assuming a perfect oracle selection mechanism. Table 7 shows that TROVE slightly outperforms PRIMITIVE in different values of $K$, mostly consistent by $1\%$.

When combining the maximum 15 answers of the 5 seeds, we can plot the performance for up to 75 samples. Figure 4 depicts how the accuracy changes for each of the approaches presented in this work. Interestingly, also for the oracle as for the one-stage agreement-based selection mechanism, TROVE slightly outperforms PRIMITIVE, for the oracle by $0.75\%$ after 75 samples and for the agreement-based method by $2.62\%$. However, for the agreement-based selection modes, the performance does not anymore increase significantly for TROVE nor for PRIMITIVE. Therefore, to leverage the full potential of both approaches, it may be worth researching more evolved selection mechanisms.

Table 7: Primitive (P) and Trove (T) $pass@K$ values (using the oracle selection mechanism) for different values of $K$. We report the mean value across 5 random seeds.

| Category | P@1 | T@1 | P@3 | T@3 | P@6 | T@6 | P@9 | T@9 | P@12 | T@12 | P@15 | T@15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algebra | 0.13 | — | 0.24 | **0.29** | 0.32 | **0.36** | 0.38 | **0.41** | 0.41 | **0.44** | 0.44 | **0.47** |
| Counting | 0.16 | — | 0.27 | **0.28** | 0.34 | **0.35** | 0.38 | **0.40** | 0.42 | **0.43** | 0.44 | **0.46** |
| Geometry | 0.05 | — | **0.11** | 0.09 | **0.18** | 0.16 | **0.22** | 0.19 | **0.25** | 0.22 | **0.28** | 0.24 |
| Intermediate | 0.07 | — | **0.14** | 0.12 | **0.19** | 0.18 | **0.23** | 0.22 | **0.25** | 0.25 | **0.28** | 0.27 |
| Number | 0.20 | — | **0.31** | 0.31 | **0.40** | 0.39 | 0.44 | **0.45** | 0.47 | **0.48** | 0.50 | **0.52** |
| Prealgebra | 0.20 | — | 0.35 | **0.36** | 0.44 | **0.45** | 0.49 | **0.50** | 0.53 | **0.54** | 0.56 | **0.57** |
| Precalculus | 0.06 | — | 0.12 | **0.17** | 0.19 | **0.23** | 0.22 | **0.26** | 0.26 | **0.29** | 0.29 | **0.31** |
| MATH | 0.14 | — | 0.24 | **0.26** | 0.32 | **0.33** | 0.37 | **0.38** | 0.40 | **0.41** | 0.43 | **0.44** |

### B.5. Distribution among MATH difficulties

Each task in MATH is further labeled with a difficulty level from 1 (easiest) to 5 (most difficult), allowing for fine-grained analysis of model performance across both topic and complexity. Table 8 provides a breakdown of the dataset by category and difficulty level.

We analyze whether TROVE differs from PRIMITIVE in the distribution of solved difficulties. For this, the oracle mechanism is chosen again to evaluate the potential of both TROVE and PRIMITIVE. Table 9 shows that both approaches solve a higher proportion of simple tasks than difficult ones. While they find a solution in more than $70\%$ of the tasks of level 1, at level 5 they only solve roughly $20\%$. Although TROVE solves slightly more tasks at difficulty level 5, again there is no

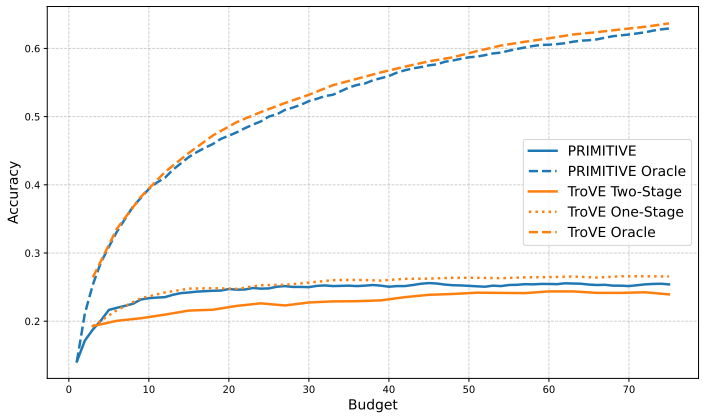


Figure 4: Accuracies for PRIMITIVE and TROVE across different selection mechanisms up to a sample size of 75. TROVE Two-Stage refers to the original implementation as discussed in Section 4.1.2, TROVE One-Stage for the corrected mechanism.

Table 8: Distribution of problems in the MATH dataset by category and difficulty level (L1–L5).

| Category | Size | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|---|
| Algebra | 881 | 125 | 159 | 192 | 209 | 196 |
| Counting | 291 | 31 | 77 | 59 | 69 | 55 |
| Geometry | 237 | 29 | 45 | 50 | 63 | 50 |
| Intermediate | 503 | 28 | 68 | 116 | 146 | 145 |
| Number | 497 | 29 | 81 | 112 | 132 | 143 |
| Prealgebra | 636 | 61 | 149 | 178 | 124 | 124 |
| Precalculus | 156 | 29 | 29 | 34 | 28 | 36 |

significant difference between both approaches. This suggests that TROVE's prompt diversity does not positively influence the performance on harder tasks.

Table 9: Analysis of the difficulty levels solved by TROVE and PRIMITIVE (in %) for $pass@K$ with $K = 15$ on the MATH dataset. The mean value across 5 random seeds is reported.

| Category | Method | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|---|
| Algebra | Primitive | 75.52 | 59.37 | 47.92 | 31.96 | 19.18 |
| | Trove | 84.00 | 65.28 | 52.08 | 33.40 | 18.78 |
| Counting | Primitive | 76.77 | 63.12 | 46.78 | 31.30 | 13.45 |
| | Trove | 79.35 | 63.64 | 52.88 | 30.14 | 15.64 |
| Geometry | Primitive | 38.62 | 44.44 | 25.20 | 27.30 | 10.80 |
| | Trove | 35.86 | 38.22 | 24.00 | 22.54 | 7.60 |
| Intermediate | Primitive | 60.00 | 46.47 | 35.00 | 23.01 | 13.79 |
| | Trove | 64.29 | 48.82 | 35.00 | 20.68 | 10.21 |
| Number | Primitive | 85.52 | 68.15 | 57.50 | 45.45 | 30.35 |
| | Trove | 83.45 | 66.91 | 57.14 | 47.12 | 36.50 |
| Prealgebra | Primitive | 79.02 | 71.68 | 55.17 | 51.29 | 31.94 |
| | Trove | 80.98 | 70.47 | 58.31 | 52.90 | 32.74 |
| Precalculus | Primitive | 64.14 | 41.38 | 18.82 | 19.29 | 10.00 |
| | Trove | 71.72 | 33.79 | 20.00 | 29.29 | 8.33 |
| Math | Primitive | 71.63 | 60.63 | 46.13 | 34.79 | 20.96 |
| | Trove | 76.02 | 61.22 | 48.37 | 35.15 | 21.34 |

### B.6. Tool Reuse

As this work complements the work by Berlot-Attwell et al. (2024), it focuses solely on the quantitative re-evaluation of TROVE. However, Berlot-Attwell et al. (2024) perform a deeper analysis on tool reuse on MATH, across all seven categories. Their findings show that TROVE only learns functions for 3 of the 7 areas — counting, number, and prealgebra — resulting in a total of just 15 learned functions. No functions are learned for algebra, geometry, intermediate algebra, or pre-calculus. Moreover, only two of these functions are ever reused in a correct solution: `is_perfect_square(n)` once and `is_prime(num)` twice. With just three reuses across 3,201 test questions, the authors conclude that TROVE's performance gains are unlikely to stem from tool reuse, which is elaborated in our work by showing that the benefit simply comes from increased compute.

### B.7. Analysis on Other Domains

To extend our evaluation to additional domains, we test the PRIMITIVE baseline on three more datasets: TabMWP (Lu et al., 2022), WTQ (Pasupat & Liang, 2015), and HiTab (Cheng et al., 2021). We run experiments using a single random seed and three different computational budgets ($K \in 1, 5, 15$). As shown in Figure 10, when compute-matched with $K = 15$, PRIMITIVE performs much more similarly to TROVE than suggested by the originally reported results.

On WTQ and HiTab, the performance of PRIMITIVE improves by approximately 7 percentage points. On TabMWP, performance increases by 8 percentage points, although in this case, the original paper's reported result already closely matches our $K = 15$ reproduction.

To validate these trends more robustly, we propose evaluating across multiple random seeds. Nonetheless, these preliminary results suggest that our core observations from MATH may generalize to other domains as well, indicating that the performance gap between TROVE and PRIMITIVE may be marginal when computational budgets are matched.

| Method | TabMWP | WTQ | HiTab |
|---|---|---|---|
| Primitive $K = 1$ | 0.36 | 0.18 | 0.08 |
| Primitive $K = 5$ | 0.42 | 0.23 | 0.14 |
| Primitive $K = 15$ | **0.44** | **0.25** | **0.15** |
| Primitive (Wang et al., 2024b) | 0.43 | 0.20 | 0.09 |
| TROVE (Wang et al., 2024b) | 0.47 | 0.21 | 0.18 |

Table 10: Accuracy across further datasets. Results show that with matched compute ($K = 15$), PRIMITIVE approaches the performance of TROVE.