TD-GROKKING: LEARNING FROM ZERO-REWARD PROBLEMS BY TRAINING-TIME DECOMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) has advanced LLM reasoning, but it cannot learn directly from difficult problems for which every sampled solution fails: their uniformly zero rewards provide no contrastive training signal. We propose TD-Grokking, a training-time decomposition framework for learning from these zero-reward problems. Guided by a reference solution, it rewrites the local reasoning requirements of each original problem as self-contained subproblems with verifiable answers. Joint training on the original problems and their subproblems allows the model to learn from verifiable local signals while continuing to practice complete solutions. Across all 15 model–benchmark combinations spanning three model families and five mathematics and science reasoning benchmarks, TD-Grokking outperforms vanilla GRPO by 4.30% on average. It also consistently improves performance across four medical question-answering benchmarks. Further analysis shows that training on subproblems alone enables 77 of 507 previously zero-reward original problems to produce at least one verified correct solution, despite never training directly on those problems. These results support training-time decomposition as a practical way to obtain useful outcome-reward signals from difficult reasoning tasks. Our code and datasets are available at `https://anonymous.4open.science/r/TD-Grokking-6567/`.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable progress in mathematical and algorithmic reasoning, with performance continuing to advance rapidly following elaborate post-training paradigms (Shao et al., 2024; Guo et al., 2025; Yang et al., 2025). As an essential component of post-training, reinforcement learning with verifiable rewards (RLVR) unlocks the inherent reasoning capabilities of LLMs by contrasting successful and unsuccessful reasoning trajectories (Shao et al., 2024; Guo et al., 2025). This naturally raises a fundamental research question: how can LLMs acquire the ability to tackle highly challenging problems where no successful trials are available? Learning from such zero-reward problems poses a critical bottleneck: training on these questions provides no outcome-reward contrast, as all sampled rollouts yield uniformly failed outcomes. For group-relative optimization, an all-fail rollout group provides no outcome-reward contrast with which to improve the policy on that question.

Prior work addresses sparse outcome rewards through process supervision, partial-credit rewards, or solution hints. Process reward models score intermediate reasoning steps (Lightman et al., 2024), while partial-credit rewards can exploit task-specific checks such as subsets of test cases (Sun et al., 2026). These approaches require intermediate supervision or checks beyond the final answer. Hint-based methods instead condition training rollouts on partial solutions or other cues to increase the chance of a successful completion (Li et al., 2026; Zhang et al., 2026; Chen et al., 2026; Liao et al., 2026; Xia et al., 2026). This raises a complementary question: can practicing local reasoning requirements as independent, verifiable tasks improve the model's ability to solve the original problem without hints?

In this work, we propose **TD-Grokking**, a training-time decomposition framework for learning from zero-reward problems. We refer to each difficult original problem as a *root problem*. A root problem may remain unsolved because the current policy cannot reliably compose several reasoning operations, even when some of those operations are within reach in isolation. Guided by

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

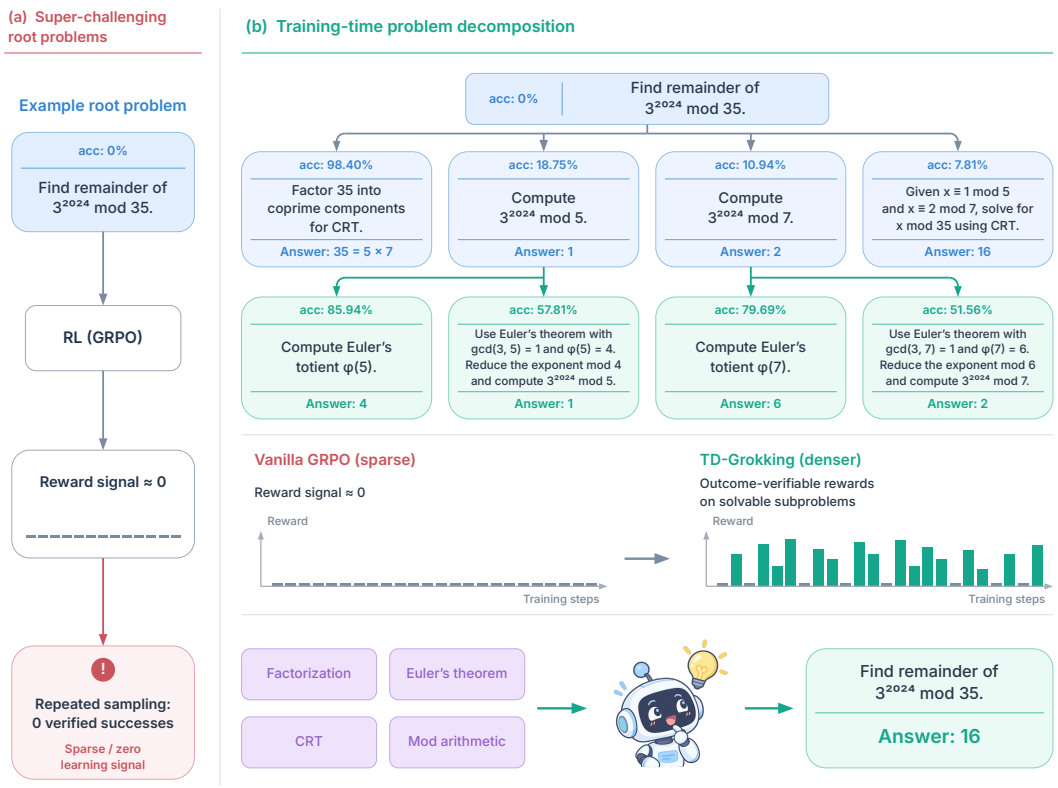


Figure 1: Overview of TD-Grokking. (a) When all sampled solutions to a difficult root problem fail, vanilla GRPO receives uniform zero rewards and therefore no outcome-reward contrast. (b) TD-Grokking decomposes the root problem into self-contained subproblems with verifiable answers, creating denser learning signals for the underlying reasoning skills.

a reference solution, TD-Grokking identifies these local reasoning requirements and rewrites them as self-contained subproblems with verifiable answers. Each subproblem becomes an independent training task, creating new opportunities for informative outcome-reward signals. We then jointly train on the root problems and their subproblems using standard GRPO and final-answer rewards. At inference time, the model receives only the original question, without subproblems or other scaffolding. Figure 1 summarizes the framework.

We evaluate TD-Grokking on mathematical reasoning and medical question answering. On Qwen3-1.7B, it raises AIME 2024 and 2025 avg@8 accuracy from 42.19% to 48.33% and from 36.67% to 40.42%, respectively. Averaged over five mathematics and science benchmarks, its gains over vanilla GRPO are 4.94%, 4.25%, and 3.72% for Qwen3-1.7B, Phi-4-mini-reasoning, and Llama-3.2-3B, respectively, yielding an overall average gain of 4.30 points across 15 model–benchmark combinations. On four medical benchmarks, TD-Grokking raises the average score from 43.57% to 46.78%, including a 6.2-point gain on PubMedQA. To isolate the contribution of subproblem practice, we train exclusively on subproblems. In a paired diagnostic, this sub-only training recovers 77 of 507 previously zero-reward roots without directly training on them, providing direct evidence that local practice can transfer to the original problems.

2 Related Work

We now review previous efforts to help LLMs tackle hard problems. We start with inference-time boosting of LLMs, and then discuss two major approaches to learning to solve hard problems.

Inference-time problem decomposition. Decomposition has also been widely used at inference time. Chain-of-thought prompting elicits intermediate reasoning steps (Wei et al., 2022). Least-to-

most prompting, decomposed prompting, Self-Ask, and Plan-and-Solve further structure inference through subproblems, plans, modular prompts, or tool-mediated steps (Zhou et al., 2023; Khot et al., 2023; Press et al., 2023; Wang et al., 2023). Tree-of-thought methods extend this idea by searching over multiple intermediate reasoning states with lookahead, backtracking, or self-evaluation (Yao et al., 2023).

These methods use decomposition as an inference-time scaffold for a fixed model. They structure the reasoning process at evaluation time. Our experiments instead examine whether subproblem practice during training improves answers to original questions. Our setting is different: the initial model receives nearly zero verified reward on the root problems. We therefore use decomposition as a training-time mechanism, where each subproblem becomes an independent RL instance rather than an intermediate prompt within a single inference attempt.

Process-level and partial feedback. When a question is so hard that the LLM cannot reach a correct answer after multiple trials, the reward will be zero for all generations, which removes the outcome-reward contrast in that sampled group. The first approach to mitigate this is to give rewards to partially correct solutions. For example, process supervision trains reward models to score intermediate reasoning steps instead of only final answers, and has been shown to improve mathematical reasoning and verification (Lightman et al., 2024). Follow-up work reduces the annotation cost by automatically constructing step-level labels or collecting process reward data through search, as in Math-Shepherd and OmegaPRM (Wang et al., 2024; Luo et al., 2024). Using a separate process reward model introduces training and validation costs, and its reliability on the target reasoning tasks requires evaluation (Zheng et al., 2025). In coding tasks, RL-Grok uses partial-correctness signals from subsets of test cases and reports a delayed increase in full-pass reward (Sun et al., 2026). Such feedback depends on task-specific intermediate checks.

Training-time hints and scaffolded exploration. A related line of work improves exploration by making difficult problems easier during training. QuestA augments hard questions with partial solution sketches (Li et al., 2026); Scaf-GRPO injects hierarchical hints when GRPO encounters all-fail rollout groups (Zhang et al., 2026); NuRL and self-hinting methods use generated cues to move hard prompts into a learnable region (Chen et al., 2026; Liao et al., 2026); and HiLL studies whether hinted success transfers to the no-hint setting (Xia et al., 2026).

These methods study how hints can make successful training rollouts more accessible. Our method constructs independent subproblem prompts with their own verifiable answers, then trains on those prompts alongside the original questions. This uses reference-solution information to define local training tasks, with no hints supplied at evaluation time.

3 METHOD

A root problem may yield no successful rollouts even when some of its local reasoning requirements are within reach of the policy model. TD-Grokking converts these local requirements into self-contained subproblems with independently verifiable answers. Joint training on roots and subproblems provides outcome-reward signals for local reasoning while retaining practice on complete solutions. TD-Grokking changes the training tasks rather than the optimizer: it specifies how roots and subproblems are constructed and combined, while standard GRPO performs the optimization. Figure 2 summarizes the five-stage construction pipeline, from selecting zero-reward roots to producing an aligned corpus of root problems and subproblems for joint training.

3.1 PROBLEM SETTING

Let x denote a reasoning problem with ground-truth verifiable answer y. A policy model π_θ generates a solution trajectory $o \sim \pi_\theta(\cdot \mid x)$, from which an answer $\hat{y}(o)$ is extracted. The standard outcome reward is

$$R(x, o) = \mathbf{1}\left[\mathrm{Verify}\big(\hat{y}(o), y\big) = 1\right].$$

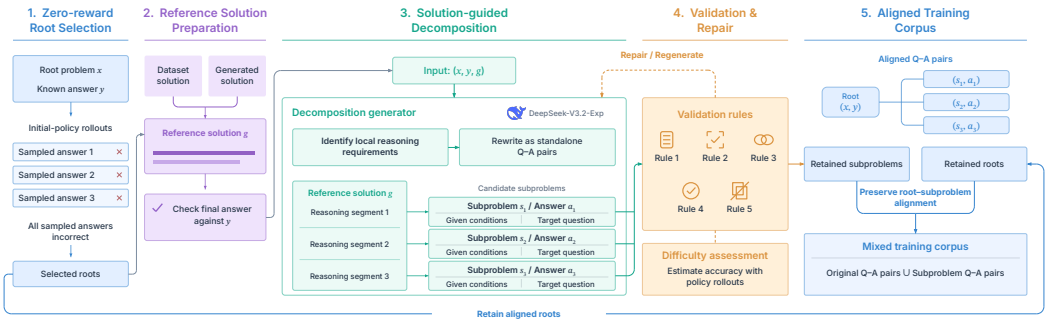


Figure 2: Training-data construction in TD-Grokking. Zero-reward roots are decomposed into self-contained subproblems using answer-consistent reference solutions. After validation and repair, retained roots and subproblems form an aligned corpus for joint GRPO training. Decomposition is used only during training-data construction.

We define the empirical accuracy of x under policy π and sampling budget K as

$$\mathrm{Acc}_K^{\pi}(x) = \frac{1}{K}\sum_{k=1}^{K} \mathbf{1}\left[\mathrm{Verify}\big(\hat{y}(o_k), y\big) = 1\right].$$

For an initial policy π_{θ_0}, a root problem x is called *zero-reward* under budget K if

$$\mathrm{Acc}_K^{\pi_{\theta_0}}(x) = 0.$$

This definition is policy- and budget-dependent: the problem is not assumed to be intrinsically unsolvable, but the sampled rollout group provides no outcome-reward contrast for direct GRPO.

3.2 Root-Problem Selection and Reference-Solution Preparation

As illustrated in Figure 2, the first stage of our construction pipeline selects root problems with zero verified accuracy under the initial policy and construction sampling budget. For each selected root x with known answer y, we then prepare an answer-consistent reference solution g. When a dataset-provided solution is available, we retain it after verifying its consistency with y; otherwise, a stronger external model generates a solution sketch whose final answer must verify against y. Decomposition therefore operates on (x, y, g), using the reference solution to identify the local reasoning requirements from which subproblems are constructed.

3.3 Constructing Self-Contained Subproblems

The decomposition stage (stage 3) transforms local reasoning requirements into independently solvable tasks with verifiable answers. Given (x, y, g), the generator identifies a short, ordered list of requirements in the reference solution, such as deriving a recurrence, resolving a case split, or computing an intermediate quantity. It then extracts the conditions required for each step and rewrites the step as a problem–answer pair:

$$\mathcal{D}(x, y, g) = \{(s_{x,1}, a_{x,1}), \ldots, (s_{x,m_x}, a_{x,m_x})\},$$

where $s_{x,j}$ denotes a subproblem and $a_{x,j}$ its target answer.

Each subproblem contains all information needed for an independent solution. When necessary, intermediate facts from the reference solution are stated explicitly as given conditions, while the target conclusion is left for the policy to derive. Consequently, a subproblem can be solved without access to the root solution or any other subproblem, and its answer can be checked using final-answer verification. Specifically, we construct the subproblems using few-shot prompting with DeepSeek-V3.2-Exp. Generated problem–answer pairs are parsed and validated, then repaired or regenerated when necessary before entering the training pool. Appendix A provides the complete prompt templates and generation settings.

3.4 Validation and Difficulty Assessment

The validation stage (stage 4) retains candidates that satisfy three criteria: they capture reasoning requirements from the reference solution, are self-contained, and have answers compatible with the outcome verifier. We check that the question and answer fields are parseable, all necessary assumptions are stated explicitly, and no candidate refers to another subproblem. We also filter excessive duplication, direct copies of the root problem, and prompts that reveal the target answer. If too few candidates pass validation or important reasoning requirements are missing, we repeat construction with stricter instructions. The retained problem–answer pairs form the subproblem training pool.

During data construction, we estimate the difficulty of each subproblem $s_{x,j}$ using $\mathrm{Acc}_{K_c}^{\pi_{\theta_0}}(s_{x,j})$, where K_c denotes the construction-time rollout budget. A nonzero estimate indicates that the initial policy produces at least one verified correct answer under this budget. These estimates characterize the outcome-reward signal available from the constructed subproblems. Appendix A reports the sampling protocol and corpus accounting.

3.5 Joint Training on Roots and Subproblems

The final construction stage combines retained roots and their derived subproblems into an aligned training corpus. TD-Grokking then jointly trains on this corpus. During training, problems are sampled from their union, and each problem is treated as an independent instance with its own target answer and standard final-answer reward. Root problems provide end-to-end practice on complete solutions, whereas subproblems provide targeted practice on local reasoning requirements. We also introduce a dynamic variant of TD-Grokking that adaptively activates or retires roots and subproblems based on sampled accuracy; Appendix A.5 provides the selection rule.

4 Experiments

We apply TD-Grokking to improve training on zero-reward problems in two distinct domains. On competition-level math, we test the main hypothesis: whether training-time decomposition can turn otherwise uninformative roots into useful RL training signals. We then turn to the medical domain to test whether TD-Grokking remains useful outside mathematical reasoning. Across all experiments, decomposition is performed only during training-data construction; evaluation is performed on the original benchmark questions without inference-time decomposition.

4.1 Mathematical Reasoning

Setup. Our primary mathematical experiments use Qwen3-1.7B with GRPO and a final-answer verification reward (Yang et al., 2025; Shao et al., 2024). We evaluate on AIME 2024/2025, AMC23, OlympiadBench, and GPQA-Diamond (Zhang & Math-AI Team, 2024; 2025; MAA, 2023; He et al., 2024; Hendrycks et al., 2021b; Rein et al., 2024). To assess generality across model families, we also compare the base model, vanilla GRPO, and TD-Grokking using Phi-4-mini-reasoning and Llama-3.2-3B-Instruct. The Qwen study additionally compares TD-Grokking with SFT, QuestA, and Scaf-GRPO, alongside Sub-only and Dynamic variants. Finally, **MWP-GRPO** controls for the effect of additional training data by augmenting root training with $15{,}487$ generic, medium-difficulty problems from the MWP-RLVR pool ($11{,}896$ from DOLPHIN18K-CLEAN-SINGLE-NUMERIC, $2{,}373$ from MAWPS, and $1{,}218$ from ASDIV-A (Huang et al., 2016; Koncel-Kedziorski et al., 2016; Miao et al., 2020)), closely matching the number of subproblems used by TD-Grokking. This baseline tests whether the observed gains arise merely from adding a comparable number of easier examples, rather than from using subproblems derived specifically from the target roots.

Training corpus. As illustrated in Figure 2, our construction starts from $4{,}944$ DeepScaleR-hard root problems on which the initial Qwen3-1.7B policy achieves zero verified accuracy under the construction sampling budget. After decomposition, validation, repair, and filtering, we retain $3{,}788$ roots and $14{,}717$ subproblems, averaging 3.89 subproblems per root. From this corpus, we construct the three controlled training views summarized in Table 1. Vanilla GRPO contains each retained root once, Sub-only contains all corresponding subproblems, and TD-Grokking contains the union

Table 1: Composition of the controlled training sets used in the mathematical experiments. Vanilla GRPO trains on the retained roots, Sub-only trains on their derived subproblems, and TD-Grokking trains on the union of both sets. Counts denote training question–answer pairs rather than sampled rollouts.

Training view	Root Q–A pairs	Subproblem Q–A pairs	Total
Vanilla GRPO	3,788	0	3,788
Sub-only	0	14,717	14,717
TD-Grokking	3,788	14,717	18,505

Table 2: Mathematics and science reasoning results (%). The upper block compares TD-Grokking with alternative training methods on Qwen3-1.7B; the lower blocks evaluate its generality across Phi-4-mini-reasoning and Llama-3.2-3B-Instruct. Results are reported as mean ± standard deviation across evaluation seeds, using eight seeds for AIMEs and five for all other benchmarks. Bold denotes the highest mean within each backbone.

Backbone	Training	AIME 2024	AIME 2025	Olympiad	AMC23	GPQA-Diamond
Qwen3-1.7B	Base	42.08±1.73	35.83±1.54	56.53±1.62	74.00±2.24	36.36±2.17
	SFT	43.75±2.78	37.08±2.14	57.57±1.58	74.50±2.74	33.84±1.96
	Vanilla GRPO	42.08±2.48	36.67±2.52	56.23±1.47	77.50±2.50	34.34±2.05
	MWP-GRPO	46.67±1.78	34.17±2.36	57.42±1.82	75.50±2.74	36.87±2.26
	QuestA	47.50±2.36	37.50±2.36	58.31±1.76	78.00±2.74	38.89±2.42
	Scaf-GRPO	42.08±1.73	34.58±1.73	57.86±1.39	76.00±2.24	38.38±2.34
	TD-Grokking	**48.33**±1.78	**40.42**±1.18	**59.64**±1.91	**83.50**±2.85	**39.90**±2.50
Phi-4-mini-reasoning	Base	48.33±1.78	31.67±2.52	59.05±1.87	87.50±2.50	44.55±2.12
	Vanilla GRPO	47.08±2.14	33.33±1.78	58.75±1.62	85.00±3.06	45.15±2.04
	TD-Grokking	**52.92**±2.14	**35.42**±1.73	**61.75**±1.65	**92.00**±2.09	**48.48**±1.96
Llama-3.2-3B-Instruct	Base	6.25±1.18	0.42±1.18	13.06±1.28	7.50±1.77	25.25±1.86
	Vanilla GRPO	4.58±1.73	0.83±1.54	14.63±1.34	15.00±2.50	26.26±1.89
	TD-Grokking	**8.33**±1.78	**4.58**±1.73	**17.03**±1.46	**20.50**±3.26	**29.80**±1.92

of exactly these roots and subproblems. The comparison therefore holds the underlying root pool fixed while varying whether training uses original tasks, local tasks, or both. Appendix A.1 provides additional corpus accounting.

Main results. Table 2 shows that TD-Grokking achieves the highest mean score among all baseline approaches on all five benchmarks in the Qwen3-1.7B comparison. Relative to vanilla GRPO, it improves performance by 4.94 percentage points on average, including gains of 6.25 and 3.75 points on AIME 2024 and 2025 avg@8, respectively. Notably, on GPQA-Diamond, an out-of-domain science benchmark, vanilla GRPO fails to improve upon the base model's 36.36% accuracy, whereas TD-Grokking achieves an accuracy of 39.90%, suggesting that its benefits extend beyond mathematics. The gains generalize across model families: TD-Grokking exceeds vanilla GRPO on every reported benchmark for both Phi-4-mini-reasoning and Llama-3.2-3B-Instruct, with average improvements of 4.25 and 3.72 points.

MWP-GRPO controls for the effect of adding a comparable number of generic, easier training examples, while the SFT baseline provides direct supervision using the same reference solutions that guide decomposition. TD-Grokking outperforms both controls across all five benchmarks in Table 2, indicating that its gains cannot be explained merely by additional training data or access to reference solutions.

Figure 3 (left) tracks accuracy on the original root problems during training. Vanilla GRPO remains near 5%, whereas Sub-only training raises root accuracy above 20% despite receiving no direct root-level rewards. Joint training yields the strongest improvement, reaching approximately 35%. These results provide direct evidence that practice on local subproblems transfers to the original tasks and that combining local and complete-task practice further strengthens this transfer. Table 3 reports

Table 3: Qwen3-1.7B training-set ablations and dynamic-selection results (%). AIME reports avg@8 and pass@8; all other benchmarks report pass@1. Bold denotes the highest score in each column, including ties.

Training	AIME 2024		AIME 2025		Olympiad	AMC23	GPQA-Diamond
	avg@8	pass@8	avg@8	pass@8	pass@1		
Sub-only	40.42	73.33	34.58	50.00	58.01	79.58	34.34
Root-only	42.08	76.67	36.67	56.67	56.23	77.50	34.34
Mixed (TD-Grokking)	**48.33**	**80.00**	**40.42**	60.00	**59.64**	**83.33**	**39.90**
Dynamic	47.91	**80.00**	39.58	**63.33**	58.01	80.83	38.89

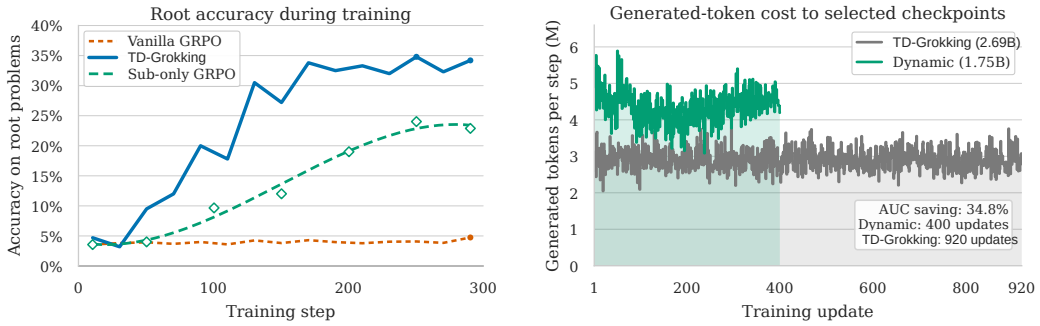


Figure 3: Training dynamics for TD-Grokking variants. Left: verified accuracy on the original root problems, averaged over 20-step windows. Right: generated response tokens per update and their cumulative area under the curve up to the selected checkpoints. Dynamic uses 1.75B generated tokens over 400 updates, compared with 2.69B over 920 updates for TD-Grokking, reducing this cost by 34.8%. This accounting excludes subproblem-construction costs.

the corresponding benchmark results, while Section 5 examines the transfer mechanism in greater detail.

Dynamic training. Dynamic uses accuracy-based task selection over the root and subproblem training pool. In Figure 3, it uses approximately 35% fewer rollout tokens than the default static problem mixing strategy. As reported in Table 3, its AIME pass@8 scores match or exceed the default setting, while its avg@8 and other reported benchmark scores are lower. This comparison describes a performance–training-token tradeoff.

4.2 Medical-Domain Instantiation

To test whether the benefits of training-time decomposition extend beyond mathematics, we apply TD-Grokking to medical question answering, a distinct domain in which difficult questions often require combining multiple local reasoning steps to reach a verifiable final answer. We train on MedBullet questions (Chen et al., 2025) and evaluate on MedQA, MedMCQA, PubMedQA, and the medical subsets of MMLU (Jin et al., 2021; Pal et al., 2022; Jin et al., 2019; Hendrycks et al., 2021a). We compare TD-Grokking with the base model, vanilla GRPO, and Sub-only training.

As shown in Table 4, TD-Grokking achieves the highest mean score on all four medical benchmarks. Compared with vanilla GRPO, it raises the average score from 43.57% to 46.78%, an improvement of 3.21 percentage points, including a 6.20-point gain on PubMedQA. Relative to the strongest control on each benchmark, the gains are 2.12 points on MedQA, 2.08 on MedMCQA, 4.10 on PubMedQA, and 2.47 on MMLU Medical. Sub-only training does not improve the overall average, indicating that combining local subproblem practice with complete-question training is particularly important in this domain. These results extend the evidence for TD-Grokking beyond mathematical reasoning.

Table 4: Medical question-answering results (%) after training on MedBullet. MMLU MEDICAL is the macro-average over five medical MMLU subsets.

Benchmark	Base	Vanilla	Sub-only	TD-Grokking
MEDQA	27.10±1.32	29.14±1.28	27.10±1.36	**31.26**±1.24
MEDMCQA	39.59±0.95	39.59±0.91	39.59±0.98	**41.67**±0.89
PUBMEDQA	54.20±1.87	52.10±1.92	52.10±1.85	**58.30**±1.78
MMLU MEDICAL	52.60±0.88	53.44±0.85	52.60±0.91	**55.91**±0.82
Average	43.37	43.57	42.85	**46.78**

5 ANALYZING TRANSFER FROM SUBPROBLEMS TO ORIGINAL PROBLEMS

We next ask whether practice on local subproblems transfers to their original root problems and how closely improvements at the two levels are related. We isolate this effect using Sub-only training, in which the policy is optimized exclusively on derived subproblems and receives no direct training signal from the roots.

Subproblem-only training recovers zero-reward roots. Figure 3 (left) suggests that training exclusively on subproblems can improve accuracy on the original roots, while Figure 4 quantifies this transfer under the diagnostic sampling budget. Before training, 507 of 514 roots have no verified successes, leaving only 1.4% with nonzero accuracy. After Sub-only training, 84 roots (16.3%) produce at least one verified correct solution. Among them, 77 are newly recovered from the original zero-reward set, corresponding to a recovery rate of $77/507 = 15.2\%$. These percentages measure nonzero-success coverage rather than mean rollout accuracy. Although most roots remain unsolved, this recovery demonstrates that local subproblem practice can transfer to complete problems without direct root training.

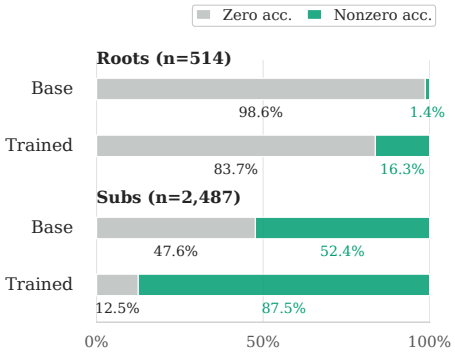


Figure 4: Composition of zero-reward problems before and after sub-only training.

Subproblem and root performance improve across skill groups. We next examine whether the observed transfer is concentrated in particular types of reasoning. We group the paired subproblems and their associated roots into 10 skill categories and compare their accuracy before and after Sub-only training. As shown in Figure 5, subproblem accuracy improves in every category, with gains ranging from 3.9 to 18.4 percentage points, while accuracy on the associated roots improves by 7.2 to 23.1 points. The largest root-level gains occur in geometry measurement, geometry relations, and equation solving.

Learning local skills alone is insufficient for most root problems. Sub-only training improves performance on local tasks, yet most associated roots remain unsolved, suggesting that learning local skills in isolation is insufficient for solving complete problems. To quantify this relationship, we correlate root accuracy with three summaries of the associated subproblem accuracies at the trained Sub-only checkpoint. Pearson correlations are 0.25 for the product, 0.27 for the minimum, and 0.23 for the log-product; the corresponding Spearman correlations are 0.28, 0.30, and 0.27. These weak positive associations indicate that local competence only partially predicts root-level success. Complete solutions likely also require selecting, ordering, and coordinating the appropriate skills, although the correlations alone cannot establish this mechanism. This observation motivates joint training on roots and subproblems, whose benefits are evaluated in Table 3.

Case studies illustrate recovery and more reliable reasoning paths. To illustrate how transfer manifests in individual problems, we examine saved diagnostic rollouts for two selected roots. The first is a constrained counting problem for which the base model produces no verified correct solution,

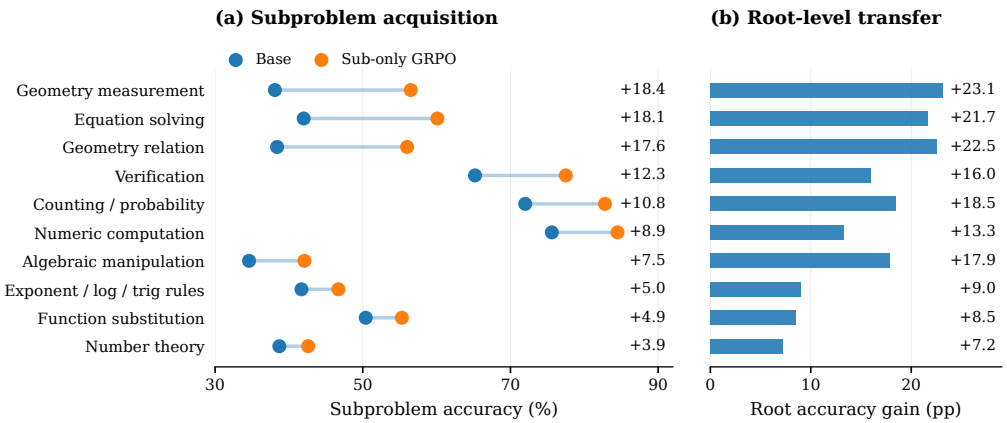


Figure 5: Accuracy changes of subproblems and root problems after sub-only RL in 10 subdomains.

Table 5: Two illustrative root-level transfer cases after sub-only training. Problem statements are condensed from the saved case records, retaining the mathematical conditions and target answers. Accuracy is estimated from 64 rollouts per model and reported in percent; these selected cases do not estimate the typical gain across roots.

Type	**Problem**	**Base**	**Sub-only**	**Gain (pp)**
Counting	Divide 15 dogs, including Rex and Daisy, into groups of sizes 6, 5, and 4, with Rex in the 6-dog group and Daisy in the 4-dog group. How many groupings are possible? Answer: 72,072.	0.0	34.4	+34.4
Triangle geometry	Triangle ABC has $AB = 13$, $BC = 14$, and $AC = 15$. Points D, E, F are the midpoints of AB, BC, AC, respectively. The circumcircles of BDE and CEF intersect at E and $X \neq E$. Find $XA + XB + XC$. Answer: $195/8$.	20.3	79.7	+59.4

but which becomes solvable after Sub-only training. The second is a geometry problem that the base model occasionally solves, but for which Sub-only training makes the successful reasoning path substantially more reliable. For each root, pass@1 is estimated from 64 rollouts using only the original question.

Table 5 presents the problem statements and measured gains. On the counting problem, accuracy rises from 0.0% to 34.4% after Sub-only training. Successful post-training rollouts consistently condition on the fixed assignments of Rex and Daisy, select the remaining dogs in each constrained group, and multiply the resulting binomial factors. On the triangle geometry problem, accuracy rises from 20.3% to 79.7%. Base-model rollouts alternate among coordinate geometry, unsupported special-point guesses, and incomplete circle arguments, whereas the trained model more consistently follows a complete route: computing the relevant midpoints and circumcircles, locating their second intersection, and obtaining the distance sum $195/8$. These cases illustrate both recovery and improved solution-path reliability. Appendix C provides additional examples.

6 Conclusion

This paper shows that TD-Grokking can make otherwise uninformative hard problems useful for post-training through training-time decomposition. Improvements across multiple model families and reasoning domains, together with transfer from subproblems to original tasks, indicate that targeted local practice can strengthen end-to-end reasoning. More broadly, it offers a promising direction for extending outcome-based learning beyond problems that are reachable by the current policy.

AI USE STATEMENT

Generative AI tools were used for training-data construction and manuscript preparation. As described in the method and Appendix A.3, DeepSeek-V3.2-Exp generated candidate subproblem–answer pairs and, when needed, reference solution sketches; these outputs underwent the validation and repair procedures described in the paper. AI assistants also helped draft and revise manuscript text, edit LaTeX, and prepare the data-construction diagram in Figure 2. The authors take responsibility for the final content of this work, including AI-assisted text and artifacts.

ETHICS STATEMENT

This work studies language-model training using existing mathematical reasoning and medical question-answering benchmarks and does not involve human-subject experiments or crowdsourcing. We credit the original models, datasets, and software and use these assets in accordance with their stated licenses and terms of use. The medical experiments assess benchmark performance only and do not establish clinical safety or suitability for patient care. Model outputs may contain factual errors or inherit biases from the underlying models and data; improved benchmark accuracy should not be interpreted as evidence of reliable medical advice. AI-generated subproblems may also contain errors despite validation, as discussed in Appendix D. Any downstream use in consequential settings would require independent validation and appropriate human oversight.

REPRODUCIBILITY STATEMENT

Appendix A documents the experimental setup, including training-corpus accounting (Appendix A.1), decomposition prompts and validation procedures (Appendix A.3), the dynamic active-set rule (Appendix A.5), benchmark and decoding protocols (Appendix A.6), and main training hyperparameters and hardware (Appendix A.7). The anonymized artifact described in Appendix A.8, available at `https://anonymous.4open.science/r/TD-Grokking-6567`, provides data-construction prompts, preprocessing scripts, training launchers, evaluation scripts, and configuration files for reproducing the main comparisons. All benchmark evaluations use the original questions without test-time decomposition. The main-table standard deviations quantify variation across evaluation seeds, rather than independent training runs; stochastic generation and differences in software or hardware may affect exact numerical replication.

REFERENCES

Hanjie Chen, Zhouxiang Fang, Yash Singla, and Mark Dredze. Benchmarking large language models on answering and explaining challenging medical questions. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3563–3599, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.182. URL https://aclanthology.org/2025.naacl-long.182/.

Justin Chih-Yao Chen, Becky Xiangyu Peng, Prafulla Kumar Choubey, Kung-Hsiang Huang, Jiaxin Zhang, Mohit Bansal, and Chien-Sheng Wu. Nudging the boundaries of llm reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hfNnQHkTtv.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand,

2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 887–896, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1084. URL https://aclanthology.org/P16-1084/.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021. doi: 10.3390/app11146421.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1259. URL https://aclanthology.org/D19-1259/.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, San Diego, California, 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136/.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. Questa: Expanding reasoning capacity in llms via question augmentation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3MifB0f7qR.

Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, and Jiang Bian. Self-hinting language models enhance reinforcement learning. *arXiv preprint arXiv:2602.03143*, 2026. URL https://arxiv.org/abs/2602.03143.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024. URL https://arxiv.org/abs/2406.06592.

MAA. American mathematics competitions. American Mathematics Competitions, 2023.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92/.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. MedMCQA: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pp. 248–260. PMLR, 2022. URL https://proceedings.mlr.press/v174/pal22a.html.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *Proceedings of the First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Rl grokking recipe: How does rl unlock and transfer new algorithms in llms? In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CJJ8VxOWbG.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Yu Xia, Canwen Xu, Zhewei Yao, Julian McAuley, and Yuxiong He. Learning to hint for reinforcement learning. *arXiv preprint arXiv:2604.00698*, 2026. URL https://arxiv.org/abs/2604.00698.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin

Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-grpo: Scaffolded group relative policy optimization for enhancing llm reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bOwVr0yr7r.

Yifan Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2024. https://huggingface.co/datasets/math-ai/aime24, 2024. URL https://huggingface.co/datasets/math-ai/aime24. Dataset.

Yifan Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2025. https://huggingface.co/datasets/math-ai/aime25, 2025. URL https://huggingface.co/datasets/math-ai/aime25. Dataset.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

A Experimental Details

This appendix describes data construction, the reported experimental settings, the archived prompts, and qualitative examples. Benchmark evaluations use the original questions. Root-level diagnostic evaluations likewise use root prompts; subproblem-level diagnostics are identified separately. Decomposition guides, associated subproblem lists, and intermediate answers are not added to root prompts at evaluation time.

A.1 Training Corpus Accounting

The mathematical training corpus is constructed from a hard subset of DeepScaleR using Qwen3-1.7B as the calibration policy. The initial pool contains $4{,}944$ root questions with no verified successful rollout under the construction-time sampling budget. This is an empirical, policy-dependent selection criterion: it does not imply that a root has zero true success probability, or that the same root is empirically zero-reward for another backbone. The construction budget is distinct from both the training group size and the 64-rollout diagnostic budget.

Decomposition, validation, and salvage yield $3{,}989$ usable roots. A subsequent solution-length filtering step, identified internally as `solq95`, leaves $3{,}788$ roots and $14{,}717$ subproblems for the main H100 runs. Thus, 955 roots are not retained in the first stage and a further 201 are not retained

Table 6: Mathematical training corpus for the reported H100 runs. Counts are question–answer records, not sampled rollouts. The joint pool contains an average of $14{,}717/3{,}788 \approx 3.89$ subproblems per root.

Training view	Records	Definition
Vanilla GRPO	3,788	Original root questions with their reference answers.
Sub-only	14,717	Self-contained subproblem–answer pairs derived from the same roots.
TD-Grokking	18,505	Union of the two views, retaining the alignment through `root_problem_id`.

after filtering. These are accounting differences, not a breakdown of the individual rejection reasons. Table 6 summarizes the resulting training views.

Each retained subproblem is associated with one root. Each retained root appears once as a root record in the Vanilla GRPO and TD-Grokking datasets. This construction fixes the underlying root pool across these three training views; it does not by itself equate their sampling frequencies or compute budgets.

A.2 Formal RLVR Signal View

Let a verifiable training item be $z = (q_z, a_z)$, with prompt q_z and reference answer a_z. For a rollout $o_{z,k} \sim \pi_\theta(\cdot \mid q_z)$, let E extract the final answer and V denote the verifier. The binary outcome reward and empirical accuracy under $K \geq 1$ rollouts are

$$r_{z,k} = \mathbf{1}[V(E(o_{z,k}), a_z) = 1], \qquad \widehat{p}_K(z;\theta) = \frac{1}{K}\sum_{k=1}^{K} r_{z,k}.$$

For a root question x with reference answer y_x, write $z_x = (x, y_x)$. It is empirically zero-reward for the initial policy under budget K when $\widehat{p}_K(z_x;\theta_0) = 0$. This describes the observed samples, not an intrinsic impossibility result or an estimate that the true success probability is exactly zero.

For a GRPO-style group of $G \geq 2$ rollouts from the same item, consider the group-relative outcome advantage with population standard deviation:

$$\bar{r}_z = \frac{1}{G}\sum_{g=1}^{G} r_{z,g}, \qquad s_z = \sqrt{\frac{1}{G}\sum_{g=1}^{G}(r_{z,g} - \bar{r}_z)^2}, \qquad A_{z,g} = \frac{r_{z,g} - \bar{r}_z}{s_z + \epsilon}, \quad \epsilon > 0.$$

If all rewards are zero, or all are one, then $s_z = 0$ and every outcome advantage is zero. This observation concerns the outcome-relative term; it does not rule out parameter updates from other prompts or other loss terms.

For G independent Bernoulli outcomes with success probability p_z,

$$P_{\text{positive}}(z) = 1 - (1 - p_z)^G, \qquad P_{\text{mixed}}(z) = 1 - p_z^G - (1 - p_z)^G.$$

The first quantity is the probability of at least one success; the second is the probability of observing both successes and failures, which is required for nonzero within-group reward variance. Roots with very small true success probability rarely produce mixed groups. A derived subproblem can provide more frequent outcome contrast when its success probability lies away from both zero and one. Decomposition does not guarantee that this condition holds for every generated subproblem.

For each retained root x, let $\mathcal{D}(x) = \mathcal{D}(x, y_x, g_x)$ abbreviate the decomposition defined in the main text, with reference guide g_x. Write $z_{x,j} = (s_{x,j}, a_{x,j})$ for its jth subproblem–answer record and define

$$\mathcal{C}(x) = \{z_{x,j} : 1 \leq j \leq m_x\}, \qquad T_x = \{z_x\} \cup \mathcal{C}(x).$$

For the set $\mathcal{X}$ of retained roots, the static training views are

$$\mathcal{S}_{\text{root}} = \{z_x : x \in \mathcal{X}\}, \qquad \mathcal{S}_{\text{sub}} = \bigcup_{x \in \mathcal{X}} \mathcal{C}(x), \qquad \mathcal{S}_{\text{mix}} = \mathcal{S}_{\text{root}} \cup \mathcal{S}_{\text{sub}}.$$

These collections refer to indexed training records: root and row identifiers preserve alignment even when two records contain identical text.

The dynamic variant maintains active records $\mathcal{A}_t \subseteq \mathcal{S}_{\text{mix}}$. Its reported activation and retirement thresholds are

$$\widehat{p}_8(z_x; \theta_t) = 0 \;\Rightarrow\; \text{activate the subproblems in } \mathcal{C}(x),$$
$$\widehat{p}_8(z; \theta_t) \geq 7/8 \;\Rightarrow\; \text{mark record } z \text{ for retirement.}$$

Only roots have associated children in this graph. With eight binary outcomes, "below 10%" means $0/8$, and "above 80%" means at least $7/8$. Appendix A.5 describes the scope of these rules.

For intuition, approximate root success by the joint success of a fixed collection of m local requirements. Under an independence approximation,

$$p_{\text{root}} \approx \prod_{j=1}^{m} p_j, \qquad \Delta \log p_{\text{root}} \approx \sum_{j=1}^{m} \Delta \log p_j.$$

The log expression requires positive probabilities and the same collection of requirements before and after training. Standalone subproblem accuracy is not necessarily the success probability of that requirement within a full solution. These expressions are diagnostic approximations, not assumptions required by the training algorithm or proofs of a transfer mechanism.

A.3 Decomposition Pipeline

Figure 2 summarizes the construction pipeline. For each selected root, a reference solution consistent with its answer is obtained or generated. Local requirements in this guide are then rewritten as standalone question–answer pairs. Candidates are screened for format, self-containment, answer compatibility, duplication, and excessive overlap with the complete root solution. Passing these checks makes an item compatible with the final-answer reward interface; it does not constitute a proof that the generated target answer is mathematically correct.

The production generator was DeepSeek-V3.2-Exp through a chat API. The retry-focused prompt version was `v6_text_blocks_retry_focus`. Table 7 records the documented settings.

Table 7: Documented decomposition-generation and validation settings. Limits for earlier passes can differ from those of the final retry pass.

Component	Setting
Generator	DeepSeek-V3.2-Exp via chat-completion API.
Output format	Natural-language text blocks; no JSON in the final production prompt.
Few-shot selection	One relevant example in the final retry pass, selected using coarse problem tags and style.
Output length	Up to 3,000 tokens in the final retry pass; earlier passes used smaller limits.
Sampling	Temperature 0.2, top-p 0.95.
Candidate count	Target 4–6 subproblems; the prompt permits 3–8. Salvage can retain longer decompositions.
Validation checks	Empty fields, cross-references, missing context, proof-only questions, final-answer mismatch, overly verbose answers, duplicate pairs, and excessive candidate counts.

Archived decomposition prompts. The listings preserve the static production prompt text, including its original wording. Angle-bracketed slots are filled with problem-specific content. Selected examples come from the few-shot library according to coarse problem tags. The `Verification` field describes self-containment; it is neither a training reward nor an independent mathematical correctness certificate.

Prompt templates for TD-Grokking subproblem generation

Listing 1: System message for decomposition generation.

```
You are a careful math curriculum designer.

Your job is to decompose one original math problem into a small set of useful training
     subproblems.

You must optimize for:
1. self-contained subproblems,
2. solvable subproblems,
3. independence between subproblems,
4. reusable mathematical skills,
5. exact-answer compatibility with programmatic evaluation.

Output only the requested text blocks.
If a later subproblem needs a quantity that could have been computed earlier, restate that
     concrete quantity directly inside the new question instead of referring to an earlier
     subproblem.
```

Listing 2: User prompt template for decomposition generation.

```
Decompose the following math problem into a small set of training subproblems.

Goal:
- produce subproblems that are independently solvable,
- each subproblem must restate all conditions it needs,
- each subproblem must train a reusable math skill from the original problem,
- the final subproblem must have the same final answer as the original answer.

Hard requirements:
1. Return between 3 and 8 subproblems.
2. Every subproblem must be self-contained. Do not refer to other subproblems, previous
     results, or hidden context.
3. Every subproblem must restate the numeric values, symbols, domains, constraints, and
     definitions it needs. This restatement can be natural prose or an explicit `Given:`
     clause. If a later step reuses a quantity, restate the concrete value or formula
     directly instead of citing an earlier step.
4. Do not write proof, verification, or "show that" style subproblems.
5. Do not create trivial meta subproblems such as "is the final answer correct?" or "choose
      the correct option" unless the original problem is inherently multiple-choice and that
      step still requires real reasoning.
6. Prefer concrete subproblems over placeholder-only algebra. Do not invent abstract
     variables like S, T, or R unless they are defined inside the same subproblem and
     genuinely useful.
7. Each `answer` must be a short exact final answer usable as ground truth. Do not include
     explanations, equations, or multiple sentences in `answer`.
8. Keep `reasoning`, `solution`, and `verification` concise but complete. Never leave any
     field blank. `verification` must explain why the subproblem is self-contained and
     solvable on its own, not why it matches another subproblem.
9. The last subproblem must solve for the original target and must end with the original
     answer exactly.
10. Across the whole decomposition, prefer at least two genuinely different skill steps.
     Avoid repeating the same shell sentence with only numbers changed.
11. Never return only 1 or 2 subproblems. If the problem feels simple, split it into smaller
      concrete computations anyway.
12. Do not introduce new helper objects like new polynomials, functions, sequences, points,
      or variables unless the original problem already uses them or the helper is strictly
     necessary for a short computation.
13. Do not let the first subproblem absorb the whole task. Each subproblem should ask for
     one concrete intermediate quantity, relation, or check.
14. Keep each field short, but prioritize completeness over rigid sentence counts: Question
      at most 3 sentences, Reasoning 1 to 2 short sentences, Solution 1 to 3 short sentences
     , Answer one line, Verification 1 to 2 short sentences.
15. If unsure, prefer simple arithmetic, algebraic, geometric, or probabilistic
     intermediate quantities over long theory summaries.
16. If a subproblem would otherwise be too long, shorten the decomposition or simplify the
     wording. Do not leave `solution`, `answer`, or `verification` blank.
17. Each subproblem should target exactly one concrete intermediate quantity, one concrete
     relation, or one concrete counting/probability result. Do not turn a subproblem into a
     mini-lecture or a long multi-part derivation.
18. Target 4 to 6 subproblems by default. Use 7 or 8 only when the original problem clearly
      needs them. Never exceed 8.
19. If a later subproblem needs an earlier result, rewrite the question in the form `Given <
     explicit value or formula> ...` or restate the full quantity in plain language. Never
     write phrases like `from the previous step`, `from Subproblem 3`, `using the result
     above`, or `same as before`.
```

```
20. If you are running out of space, reduce the number of subproblems instead of leaving
     the last subproblem incomplete. The final subproblem must always contain a non-empty `
     solution` and a non-empty `answer`.
21. The final `answer` should match the original answer's target and outer form as closely
     as possible. If the original answer is an expression, give only that expression. If it
     is an equation, inequality, set, ordered pair, list, or named quantity, preserve that
     outer structure instead of answering with a different but related object.
22. Do not end with a verification-only subproblem. If the last step would only check
     correctness, merge it into the previous computational step and keep the final answer
     there.
23. In `answer`, write only the target answer. Do not write explanatory prefixes such as `
     Therefore`, `So`, `The answer is`, `check`, `because`, or a full sentence.

Style preference:
<STYLE_INSTRUCTION>

Return only text in this exact structure:

### Subproblem 1
Question: ...
Reasoning: ...
Solution: ...
Answer: ...
Verification: ...

### Subproblem 2
Question: ...
Reasoning: ...
Solution: ...
Answer: ...
Verification: ...

### Subproblem 3
Question: ...
Reasoning: ...
Solution: ...
Answer: ...
Verification: ...

Use the same six labels for every subproblem block. Do not use JSON. Do not use code fences.


Good decomposition examples to imitate in spirit, not in wording:
<SELECTED_GOOD_DECOMPOSITION_EXAMPLES>

Bad decomposition pattern to avoid:
<SELECTED_BAD_DECOMPOSITION_EXAMPLES>

Original problem:
<ORIGINAL_PROBLEM>

Original answer:
<ORIGINAL_ANSWER>

Reference solution:
<REFERENCE_SOLUTION_OR_NO_REFERENCE_SOLUTION_PROVIDED>

The End of Prompt
```

For the final production retry pass, `STYLE_INSTRUCTION` was the natural-style instruction:

> Prefer natural question wording. Restate necessary conditions inside each question, but do not force every subproblem into the same explicit template. Use `Given:` only when it genuinely improves clarity.

Notes on the archived wording. Although the archived user prompt says "six labels", each output block contains a subproblem header and five named fields: `Question`, `Reasoning`, `Solution`, `Answer`, and `Verification`. The historical wording also contains tensions between the prohibition of equations in rule 7 and the answer structures allowed by rule 21, and between the ban on verification-style tasks in rule 4 and the "check" wording in rule 13. We retain these instructions to document the generation procedure, rather than retroactively present an edited prompt as the one used in the experiments. Retained outputs are assessed by the construction-time checks described above; literal prompt compliance alone does not ensure correctness.

A.4 SUBPROBLEM SKILL ANNOTATION PROMPT

After decomposition, an automated annotation pass assigns one primary skill and up to two secondary skills to each subproblem, using the frozen taxonomy below. These labels are used for analysis only, not as policy inputs, training rewards, or test-time scaffolding. The listings reproduce the annotation prompts. As model-generated analysis labels, their reliability depends on the annotation procedure; the prompt itself is not evidence of human-validated label accuracy.

Listing 3: System message for atomic skill annotation.

```
You are a careful math annotation assistant.

Your task is to classify each subproblem into an atomic skill taxonomy.

You must follow these rules strictly:

1. Use only labels from the provided label set.
2. Output exactly one primary_label.
3. Output zero to two secondary_labels.
4. Treat verification as a secondary label unless the problem is almost entirely a
     consistency check.
5. Prefer the skill required to solve the problem, not superficial wording.
6. If the problem asks for an unknown value by solving an equation or system, prefer
     equation_solving.
7. If the problem mainly asks to rewrite, simplify, or transform an expression, prefer
     algebraic_manipulation.
8. If the problem is mainly about ratios, averages, unit rates, prices, speed, or
     percentages, prefer ratio_rate_percent.
9. If the problem is mainly about counting, probability, expectation, or combinatorial
     arrangements, prefer counting_probability.
10. If the problem is mainly about divisibility, modular arithmetic, bases, remainders, gcd,
      lcm, or integer structure, prefer number_theory.
11. If the problem is mainly about geometric quantities such as length, area, volume,
     perimeter, coordinates, or chord length, prefer geometry_measurement.
12. If the problem is mainly about geometric theorems, angle relations, tangent properties,
      conics, similarity, congruence, or geometric structure, prefer geometry_relation.
13. If the problem is mainly direct evaluation of a known numeric expression, prefer
     numeric_computation.
14. If the problem is mainly about recurrence, progression, repeated updates, or iterative
     processes, prefer sequence_recurrence.
15. If the problem is mainly about function definition recovery or substitution into a
     function form, prefer function_substitution.
16. If the problem is mainly about exponent rules, logarithm rules, trigonometric
     identities, or standard trigonometric ratios/values, prefer exponent_log_trig_rules.

Output valid JSON only.

Return:
{
  "results": [
    {
      "problem_id": "...",
      "primary_label": "...",
      "secondary_labels": ["...", "..."],
      "rationale": "..."
    }
  ]
}
```

Listing 4: User prompt template for atomic skill annotation.

```
Classify the following subproblems using the fixed atomic skill taxonomy.

Allowed labels:
- numeric_computation
- algebraic_manipulation
- equation_solving
- function_substitution
- exponent_log_trig_rules
- ratio_rate_percent
- counting_probability
- number_theory
- sequence_recurrence
- geometry_measurement
- geometry_relation
- verification
```

```
Few-shot examples:
<FEWSHOT_JSON_LINES>

Target batch:
{
  "label_set": [
    "numeric_computation",
    "algebraic_manipulation",
    "equation_solving",
    "function_substitution",
    "exponent_log_trig_rules",
    "ratio_rate_percent",
    "counting_probability",
    "number_theory",
    "sequence_recurrence",
    "geometry_measurement",
    "geometry_relation",
    "verification"
  ],
  "subproblems": [
    {
      "problem_id": "...",
      "question": "..."
    }
  ]
}
```

The retained diagnostic contains 2,487 labeled subproblems associated with 514 roots. Figure 6 shows primary–secondary label co-occurrences. Each item has one primary label but may contribute to more than one co-occurrence cell, so cell counts need not sum to the number of subproblems. The taxonomy contains twelve labels. The ten categories in Figure 5 and the seven categories in Figure 7 are displayed subsets, not exhaustive partitions of the diagnostic dataset.

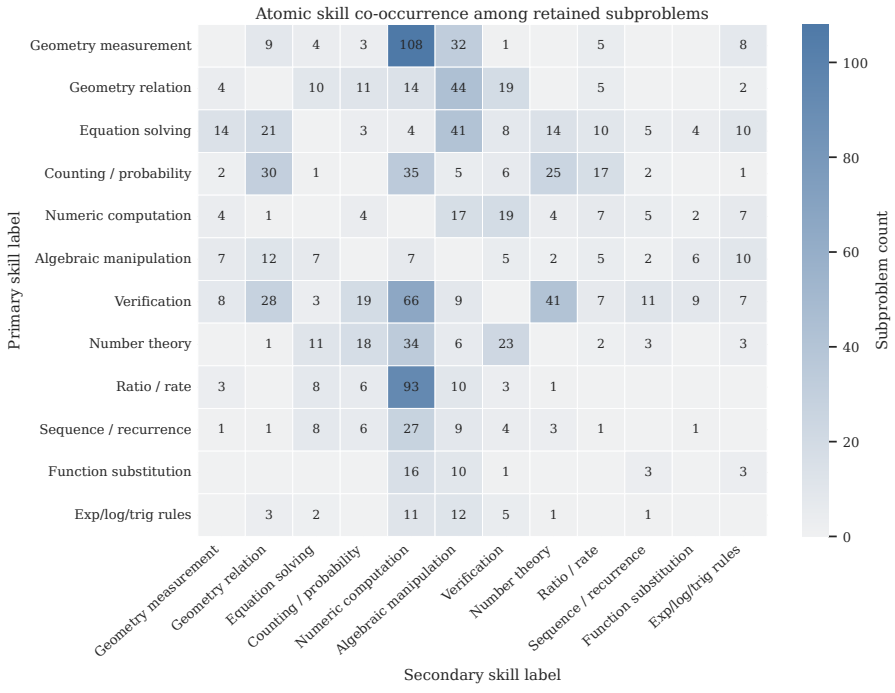


Figure 6: Primary–secondary atomic-skill co-occurrences in the paired diagnostic of 2,487 subproblems. Rows denote primary labels and columns denote secondary labels; blank cells have zero count. Co-occurrence counts are label-pair counts rather than mutually exclusive item counts.

A.5 Dynamic Active-Set Rule

The dynamic variant uses the same root–subproblem graph as the joint training view and changes the active training records. Each root starts active. A root with no successful rollout in its current group of eight activates its associated subproblems. An evaluated record with at least seven successes in eight rollouts meets the retirement threshold. A subproblem without children does not activate further tasks. These rules change the sampler without introducing a different reward function. The thresholds describe operational decisions under finite sampling, not proofs that a task is impossible or permanently mastered.

A.6 Benchmark and Generation Protocols

The mathematical and science benchmark evaluations use the local lm-eval/vLLM wrapper. For Qwen3-1.7B, the documented configuration enables the chat template and thinking mode, with temperature 0.6, top-p 0.95, top-k 20, min-p 0, and bfloat16 inference. These Qwen-specific template settings should not be interpreted as the templates for Phi-4-mini-reasoning or Llama-3.2-3B-Instruct.

For a benchmark of N items, let $c_{ik} \in \{0, 1\}$ indicate whether completion k for item i verifies. The item-level repeated-sampling metrics used for AIME are

$$\text{avg@8} = \frac{100}{8N} \sum_{i=1}^{N} \sum_{k=1}^{8} c_{ik}, \qquad \text{pass@8} = \frac{100}{N} \sum_{i=1}^{N} \mathbf{1}\left[\sum_{k=1}^{8} c_{ik} > 0\right].$$

Single-completion accuracy is $100N^{-1} \sum_i c_{i1}$. The pass@1 estimates in the case studies instead average 64 sampled completions per item to estimate single-rollout success probability. Item-level sampling budgets and independent evaluation repeats are distinct quantities. Variation across evaluation seeds is also distinct from variation across independently trained checkpoints.

Table 8: Benchmark datasets and item-level metrics. Run-specific repeat counts and aggregation conventions are separate from the sampling budgets in this table. MATH500 is an auxiliary evaluation, not a column of the main benchmark comparison.

Benchmark	Items	Metric	Protocol notes
AIME 2024	30	avg@8; pass@8	Eight completions per item. Reported main-run generation cap: 38,912 tokens. Final boxed-answer extraction and normalization.
AIME 2025	30	avg@8; pass@8	Same item-level metric definitions as AIME 2024. The ablation also reports pass@8.
OlympiadBench	674	Accuracy	Mathematical subset, using the local external-math task and final-answer equivalence checking.
AMC23	40	Accuracy	Local AMC23 task. Reported supplementary runs use a 38,912-token generation cap.
GPQA-Diamond	Diamond subset	Accuracy	Out-of-domain science benchmark in Table 2; the item-level metric is final-answer accuracy.
MATH500	500	pass@1	Auxiliary runs; reported generation caps range from 32,768 to 38,912 tokens across run families.

For the medical-domain experiments, training examples are derived from Medbullets (Chen et al., 2025). Evaluation uses generative tasks that request a final option or short answer, without the mathematical boxed-answer system instruction. Scores use task-specific normalization and exact matching. The MMLU Medical result in Table 4 is the unweighted mean over Anatomy, Clinical Knowledge, College Medicine, Medical Genetics, and Professional Medicine; it is not an item-weighted mean over their union.

The reported 1,000-item PubMedQA evaluation should not be equated with the commonly used 500-item test split. The dataset provides a 1,000-item expert-labeled collection and a splitting procedure.[1] Comparisons depend on the actual evaluation IDs and any overlap with training or model-selection data.

[1] https://github.com/pubmedqa/pubmedqa

Table 9: Reported medical evaluation datasets and scoring. Counts describe the evaluated task sets; they do not specify the number of generation repeats.

Benchmark	Items	Scoring
MedQA-USMLE	1,273	Exact match on the generated option or answer.
MedMCQA	4,183	Exact match on the generated option or answer.
PubMedQA	1,000	Exact match on normalized yes/no/maybe answers; evaluated split requires explicit identification.
MMLU Medical	Five subsets	Macro-average of the five subset accuracies listed above.

A.7 Training Hyperparameters

Table 10 records the documented settings for the main Qwen3-1.7B mathematical H100 runs. They do not by themselves specify the configurations of the Phi, Llama, medical, or earlier diagnostic runs. The outcome reward uses the mathematical final-answer verifier, with no decomposition-specific reward shaping.

Table 10: Documented settings for the main Qwen3-1.7B mathematical GRPO runs. Evaluation decoding is described in Appendix A.6.

Parameter	Value / scope
Initial checkpoint	Qwen3-1.7B (instruction-tuned checkpoint, not Qwen3-1.7B-Base).
RL algorithm	GRPO with group-relative outcome advantages.
Outcome reward	Binary final-answer verification.
Hardware	One node with 8 NVIDIA H100 GPUs.
Training batch size	64.
Rollouts per prompt	8 in the documented dynamic and later H100 runs.
Reward workers	64 CPU workers in the H100 launchers.
Response length cap	16,384 generated tokens during training.
Training sampling	Temperature 1.0, top-p 0.95, top-$k = -1$ in the filtered-corpus run family.
Rollout budget	Default budget $K = 16$, and construction-time budget $K_c = 64$.

A.8 Anonymized Artifact

The manuscript links to an anonymized code and data repository. Its intended scope is the construction prompts, preprocessing, training launchers, evaluation scripts, and run configurations. The settings described here should be interpreted together with the corresponding run records; aggregate tables alone are insufficient to reconstruct every experiment.

B Subproblem Skill Diagnostic

The paired diagnostic predates the full H100 benchmark suite and contains 514 root problems and 2,487 associated subproblems. Each item is evaluated with 64 rollouts, temperature 0.6, top-p 0.95, and a 16,384-token response cap. Root-level evaluation supplies only the root question; subproblem-only training uses derived tasks. This does not make the entire root–subproblem family a held-out test family.

Figure 4 reports the fraction of items with at least one verified success in 64 rollouts. Root coverage increases from $7/514 \approx 1.4\%$ to $84/514 \approx 16.3\%$; reported subproblem coverage increases from 52.4% to 87.5%. Coverage is not mean rollout accuracy: with the same items, checkpoint, and scoring rule, mean accuracy cannot exceed the fraction of items with any success. These diagnostic percentages should not be directly equated with training-curve accuracies evaluated under a different setup.

Figure 7 provides a descriptive view of subproblem performance. Skill categories can overlap through secondary labels, and a gain on a standalone subproblem does not establish a causal effect on its root.

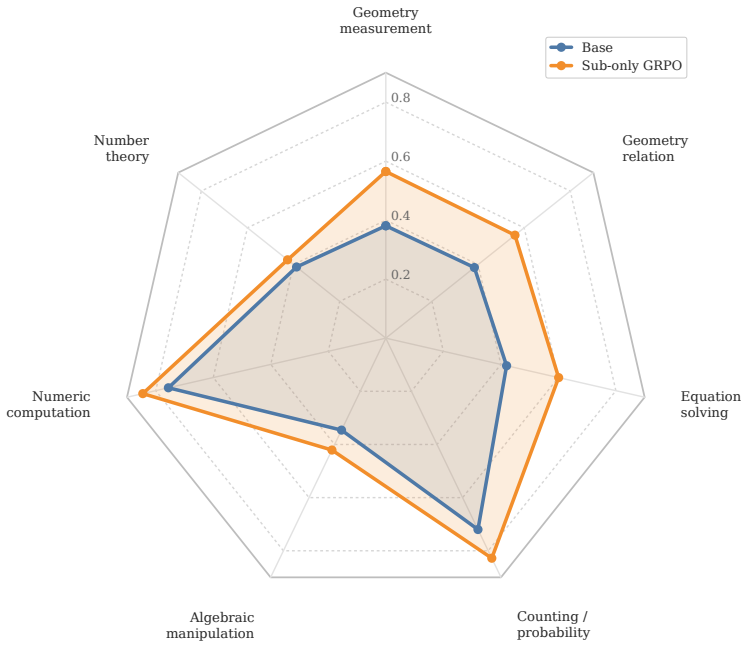


Figure 7: Subproblem accuracy for seven displayed skill categories in the paired diagnostic. Each axis compares the base model with the subproblem-only checkpoint under the same scoring convention. The displayed categories are a subset of the twelve-label taxonomy and do not provide a complete decomposition of root-level performance.

The root-level coverage diagnostic and selected saved rollouts address different questions from this skill summary.

Interpreting local-to-root correlations. For a root with associated subproblem accuracies p_{ij}, a product summary is $u_i = \prod_j p_{ij}$ and a minimum summary is $m_i = \min_j p_{ij}$. Where all probabilities are positive, the log-product is $\ell_i = \log u_i = \sum_j \log p_{ij}$. On the same observations, product and log-product have identical ranks and hence identical Spearman correlations with root accuracy. Zero handling, clipping, smoothing individual probabilities, or sample exclusion can change that relationship and must be stated explicitly. Pearson correlations need not be equal after a log transformation. These associations alone do not identify a transfer mechanism.

C Case Studies

We examine selected diagnostic roots with saved rollout text to illustrate changes in solution paths after subproblem-only training. These examples are not a random sample of roots, do not estimate a typical treatment effect, and do not isolate which individual subproblem caused a change. Table 11 retains the reported scores and distinguishes observations from input-quality limitations.

Triangle geometry. For `deepscaler_00000900`, some base-model rollouts already find a correct route, while others use unsupported special-point guesses or leave the circle calculation incomplete. A reproducible coordinate solution places $B = (0, 0)$, $C = (14, 0)$, and $A = (5, 12)$. The relevant midpoints are $D = (5/2, 6)$, $E = (7, 0)$, and $F = (19/2, 6)$. The circles have equations

$$x^2 + y^2 - 7x - \frac{33}{8}y = 0, \qquad x^2 + y^2 - 21x - \frac{33}{8}y + 98 = 0.$$

Their second intersection is $X = (7, 33/8)$, giving $XA = XB = XC = 65/8$ and $XA + XB + XC = 195/8$. The reported success counts increase from $13/64$ to $51/64$. This illustrates a more frequently completed solution route, without establishing the causal contribution of any particular training subproblem.

Table 11: Selected cases with saved rollout text. Base and Sub-only scores are pass@1 estimates in percent from 64 rollouts per model. Gain is in percentage points and is computed before rounding. Input limitations are part of the interpretation, not corrections applied retrospectively to the reported scores.

Root ID	Base (%)	Sub-only (%)	Gain (pp)	Observation / limitation
00000900	20.3	79.7	+59.4	More successful coordinate solutions to the 13–14–15 triangle problem, ending at $195/8$.
00000631	43.8	76.6	+32.8	More answers use the intended 3×3 shoe-pair sample space. The saved question contains inconsistent shoe counts.
00000312	15.6	51.6	+35.9	Successful solutions maintain the favorable count 2052 and simplify $2052/4320$ to $19/40$.
00000454	0.0	34.4	+34.4	Successful solutions condition on the fixed dog memberships and obtain $\binom{13}{5}\binom{8}{3} = 72{,}072$.
00000625	4.7	57.8	+53.1	More successful distance calculations using coordinates supplied in the problem's Asymptote code; not evidence of independent visual coordinate inference.

Counting and probability. For deepscaler_00000454, fix Rex in the six-dog group and Daisy in the four-dog group. Choose five of the other thirteen dogs to join Rex, then three of the remaining eight to join Daisy. The five unchosen dogs form the last group, giving $\binom{13}{5}\binom{8}{3} = 72{,}072$; no division by 3! is needed because the group sizes differ. For deepscaler_00000312, the selected successful solutions use 2052 favorable cases among $20 \times 6^3 = 4320$ outcomes, giving $19/40$. These computations check the stated answers, while the saved scores quantify how often the evaluator accepted the corresponding rollouts.

Input limitations in two examples. The saved text for deepscaler_00000631 describes three pairs of shoes but also refers to left and right sets of six shoes. Under the intended interpretation of three left and three right shoes, there are nine equally likely left–right combinations and three matching pairs, giving $1/3$. Because the question is internally inconsistent, this example cannot cleanly attribute all failures to a model's sample-space reasoning.

For deepscaler_00000625, the saved question includes Asymptote code specifying coordinates, including $A = (-9\sqrt{3}, 0)$ and $G = (12\sqrt{3}, 6\sqrt{6})$. Thus

$$AG = \sqrt{(21\sqrt{3})^2 + (6\sqrt{6})^2} = 9\sqrt{19}.$$

The visible successful rollouts use these supplied coordinates. They should not be described as independently recovering coordinates from an image.

Limits of the case evidence. Other saved examples contain apparent negative transfer together with equivalent-answer or possible ground-truth conflicts, including time answers expressed in different units and number-theory target disagreements. Selected positive examples therefore cannot establish the frequency of positive or negative transfer in the full corpus. The reported case scores retain the original evaluation convention; no corrected aggregate is inferred from these qualitative inspections.

D Evaluation and Data Quality Notes

Answer extraction and target quality. Mathematical outputs are scored using final-answer extraction, normalization, and equivalence checking. Exact strings, mathematically equivalent answers, and correct answers with different units can receive different outcomes if normalization is incomplete. A generated subproblem may also carry an incorrect reference answer. Format validation, final-answer matching, and teacher self-checks alone do not eliminate these possibilities. The original scorer convention is retained in the reported diagnostic scores.

Generation settings. The documented Qwen training configuration uses temperature 1.0, top-p 0.95, top-k $= -1$, and a $16{,}384$-token response cap. The Qwen benchmark configuration uses temperature 0.6, top-p 0.95, top-k 20, min-p 0, and benchmark-specific longer caps. These are distinct sampling regimes. Their settings must not be transferred to other backbones or earlier run families without checking the corresponding configurations.

No test-time decomposition and data overlap. Root and benchmark evaluations use their original questions, without added decomposition guides, derived tasks, or intermediate answers. This property does not establish that training roots or generated subproblems are disjoint from benchmark content. Without a documented overlap audit, no stronger decontamination claim follows from the training-time-only design.

AI-assisted construction and analysis. DeepSeek-V3.2-Exp generates candidate subproblems and, when needed, reference guides. An automated annotation pass supplies the diagnostic skill labels. These are distinct uses of model-generated content: the former affects training data and the latter affects qualitative and skill-conditioned analysis. Construction checks do not amount to a human correctness audit of all generated answers or labels.

Existing assets. The main Qwen backbone is documented by Yang et al. (2025). Official model cards identify Phi-4-mini-reasoning and Llama-3.2-3B-Instruct. The decomposition generator is DeepSeek-V3.2-Exp. The mathematical source corpus is based on DeepScaleR; the hard subset and filtering are specific to this study. The software stack includes verl, vLLM, and lm-evaluation-harness. These links identify upstream assets and their published documentation; they do not specify the exact revisions used in an experiment or certify license compliance for an unspecified revision.

The benchmark sources are AIME 2024/2025, AMC, OlympiadBench, MATH, and GPQA (Zhang & Math-AI Team, 2024; 2025; MAA, 2023; He et al., 2024; Hendrycks et al., 2021b; Rein et al., 2024). Medical assets are Medbullets, MedQA, MedMCQA, PubMedQA, and the relevant MMLU subsets (Chen et al., 2025; Jin et al., 2021; Pal et al., 2022; Jin et al., 2019; Hendrycks et al., 2021a).