BEYOND REPEATED SAMPLING: LEARNING SEARCH POLICIES FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly tackle hard reasoning problems by spending more test-time compute, yet the dominant strategy remains naive repeated sampling: draw many independent solutions and hope one is correct. Because such sampling explores only through local decoding noise, it tends to produce many near duplicate attempts rather than genuinely different ideas. We ask whether exploration can instead be steered at a semantic level, by first sampling problem specific concepts, hints, or strategies and then conditioning answer generation on them. We refine this into a simple, more exploratory procedure that emits many diverse concepts in a single trajectory, and evaluate it on hard problems where repeated sampling struggles. We then go a step further and make concept generation *trainable*: a small concept generator is optimized with reinforcement learning so that its concepts maximize the downstream success of a larger, frozen answer generator. On hard mathematical reasoning problems, the trained concept generator substantially improves the answer generator's pass@k over naive repeated sampling at the same answer generation allocation, surpasses concepts drawn from much larger untuned models, and transfers to answer generators it was never trained against, including a model from a different family. A small model can thus be trained into an effective, reusable search policy for a much larger one.

1 INTRODUCTION

Reinforcement learning has become a central ingredient in training modern large language models (LLMs), particularly for reasoning: a model is improved by sampling its own solutions, scoring them against a verifier or reward, and reinforcing the successful ones (Shao et al., 2024; Olmo et al., 2025). The effectiveness of this loop hinges on exploration. RL can only reinforce, and ultimately distill into the model weights, behaviors that the sampling policy manages to discover in the first place. On a hard problem where the model never stumbles on a correct solution, there is simply no positive signal to learn from. The same bottleneck appears at inference time, where discovering a correct solution among many samples is precisely what test-time scaling relies on (Snell et al., 2024). In both regimes, progress is limited by the quality of the underlying search policy, which makes the ability to explore the solution space effectively a first class concern rather than an implementation detail.

In practice, the dominant form of exploration is repeated sampling: draw many complete answers from the same model and keep those a verifier or judge deems correct (Brown et al., 2024; Wang et al., 2022). This explores only indirectly, through local token level decoding noise, and often yields many near duplicate attempts rather than genuinely different ideas. A natural alternative is to separate deciding how to approach a problem from carrying out the solution: first sample high level concepts, hints, theorems, or strategies, and then generate answers conditioned on them. This mirrors how people tackle hard problems, pausing to consider which ideas or tools might apply before committing to a full derivation, diversifying reasoning at a semantic level rather than at the level of individual tokens.

Recent work instantiated this idea as a concept guided sampling pipeline, using one LLM to propose concepts and another to answer conditioned on them (Handa et al., 2026). In our reproduction, however, its advantage largely vanishes once the repeated sampling baseline is allowed to use exploratory decoding parameters instead of the restrictive ones originally used. We further identify

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.

two practical limitations: concepts are produced one at a time in an iterative loop, which is wasteful given that LLMs are autoregressive, and only about one concept is generated per problem on average, sharply limiting the diversity the method was meant to provide.

In this paper we revisit concept guided exploration for mathematical reasoning and turn it into a trainable component. We first strengthen the inference time procedure: we generate all concepts in a single trajectory, prompt for detailed and problem specific concepts, and evaluate on hard subsets where naive repeated sampling fails outright. Building on this, we ask whether a small concept generator can be *trained*, rather than merely prompted, to produce concepts that improve a larger, frozen answer generator that may be impractical to train, or even closed-source. Concretely, we cast exploration itself as an RL problem, but over concepts rather than answers: we train the concept generator with reinforcement learning, rewarding each concept trajectory by the downstream success of the answer generator it steers. On a held-out set of hard problems, this doubles the answer generator's pass@128, from 19.0% for naive repeated sampling to 39.2% at the same answer generation allocation. Moreover, the learned policy behaves as a reusable search strategy rather than a scale effect: a trained 7B concept generator steers a 32B answer generator well enough to surpass the concepts that the larger model produces for itself, and it transfers to a 70B answer generator from a different family that it was never trained against. Our contributions are:

- Re-evaluating GuidedSampling across all 25 Qwen2.5 CG/AG size pairs (1.5B–32B) on MATH500, we find that its gains largely disappear against an exploration-enabled repeated sampling baseline.
- We introduce single trajectory concept generation and RL-train Qwen2.5-7B as a search policy for a frozen Qwen2.5-32B answer generator, raising pass@128 from 19.0% to 39.2% on 1k held-out hard DeepMath problems.
- The trained 7B CG outperforms an untuned 32B CG and transfers without retraining to Llama-3.3-70B; its gains also persist out of distribution on Omni-MATH 2.
- We analyze whether the learned generator improves exploration rather than merely solving problems directly, and study answer leakage, concept relevance, position effects, and the effect of the number of concepts.

2 Related Work

Inference-Time Search and Concept-Guided Exploration Repeated sampling is a standard way to spend inference-time compute on reasoning, with performance scaling in the number of samples (Brown et al., 2024; Snell et al., 2024) and typically paired with self-consistency or verifier-based selection (Wang et al., 2022; Lightman et al., 2024). A complementary line of work decomposes reasoning into intermediate artifacts (plans, hints, or solution sketches) before producing an answer (Wang et al., 2025). Our direct predecessor is the GuidedSampling method of Handa et al. (2026), which samples concepts from one LLM and conditions a second LLM's answers on them; unlike them, we find in our reproduction that this advantage largely disappears once the repeated-sampling baseline is allowed to explore (Section 3), and we depart from their setup by emitting all concepts in a single trajectory and by training the concept generator for downstream answer generator success rather than using it zero-shot. This also contrasts with tree-search methods such as Tree of Thoughts (Yao et al., 2023), whose dynamic branching disrupts continuous batching and KV-cache sharing in optimized engines like vLLM (Kwon et al., 2023), whereas our single trajectory generation diversifies exploration while preserving standard autoregressive throughput.

RL-Driven Abstractions and Small-Model Orchestration The closest RL-based method is RLAD (Qu et al., 2025), which trains an abstraction generator to produce procedural summaries that guide a downstream solver. Unlike RLAD, our method uses no SFT warm start based on traces from a stronger teacher, targets concise, problem-specific exploration hints rather than full procedural summaries, and trains only the concept generator. Like SOAR (Sundaram et al., 2026), we reward the generator using measured downstream performance rather than an intrinsic proxy, however, our downstream model remains frozen. Related zero-shot methods elicit similar artifacts without training the generator: Step-Back prompting abstracts a problem to higher-level principles (Zheng et al., 2024), and Auto-CoT synthesizes its own few-shot exemplars (Zhang et al., 2022) (whose gains may stem from problem creation rather than the examples themselves (Gwak et al.,

2026)). In contrast, we *train* the concept generator with trajectory-level rewards from downstream answer success. Finally, our setup echoes small-model orchestration of larger systems: Directional Stimulus Prompting (Li et al., 2023) trains a small LM to emit hints that guide a frozen black-box LLM, and ToolOrchestra (Su et al., 2025) RL-trains a lightweight orchestrator to coordinate tools and frontier models for efficiency, whereas we steer a single fixed, larger answer generator through concepts, with a trained 7B concept generator improving a 32B answer generator.

3 Background and Analysis

The method most closely related to ours first samples concepts with one LLM and then uses a second LLM to generate final answers conditioned on those concepts (Handa et al., 2026). The intended benefit is to explore diverse mathematical ideas before spending the larger answer-generation allocation. In principle, this can produce more semantically diverse rollouts than repeated sampling from the answer generator alone.

Reproduction and Baseline Sensitivity. We reproduce the concept-guided sampling method on MATH500 (Hendrycks et al., 2021; Lightman et al., 2024), using Qwen2.5-Instruct models at five sizes (1.5B, 3B, 7B, 14B, and 32B) as both the concept generator (CG) and the answer generator (AG), and evaluate all 25 CG/AG size combinations. Concepts are produced with the original iterative generation procedure and prompts from Handa et al. (2026), left unchanged. For each problem we draw a total of 100 answer rollouts and report pass@50: the baseline draws all 100 rollouts directly from the AG, whereas the concept-guided condition splits the same 100-rollout allocation across the concepts parsed from the CG trajectory, so the two conditions are matched in answer-generation compute. Only the AG sampling parameters differ from the original protocol.

The reported advantage of concept-guided sampling is highly sensitive to these parameters. The original repeated-sampling baseline used temperature 0.8 and top-$p = 0.5$, which restricts exploration and yields an artificially weak baseline. When both the baseline and the concept-conditioned answer generator are instead sampled with exploratory parameters (temperature 1.0, top-$p = 0.95$, top-k disabled, implemented as -1), the concept-guided method no longer improves over repeated sampling and slightly degrades it across nearly all CG/AG pairs. Table 1 reports, for each AG size, the baseline pass@50 (top row) and the pass@50 difference between the concept-guided method and this baseline for every CG (rows) / AG (columns) pair, where negative values indicate underperformance; the final column gives the average number of concepts generated per problem by each CG. Once the baseline is allowed to explore, the gains attributed to concept guidance largely disappear.

Table 1: Reproduction of concept-guided sampling (Handa et al., 2026) on MATH500 (Qwen2.5-Instruct): baseline pass@50 and concept-guided differences across CG/AG size pairs. Reported gains disappear once the answer generator is allowed to explore freely.

	AG					Avg Concepts
	1.5B	3B	7B	14B	32B	
Baseline pass@50	89.2	93.8	95.9	96.0	96.2	-
CG / AG	**Difference (Concept - Baseline) pass@50**					
1.5B	-0.6	-0.7	-1.0	-0.8	-0.8	1.15
3B	-1.7	-0.6	-1.3	-0.8	-1.0	1.17
7B	-1.8	+0.1	-1.8	-1.2	-0.7	1.02
14B	-1.0	-0.7	-2.2	-1.1	-1.8	1.25
32B	-1.4	-0.1	-0.8	-0.2	-0.5	1.68

We also identify two structural limitations. First, concepts are generated one by one in an iterative loop. This is slow and unnecessary because LLMs are autoregressive and will generate newer concepts conditioned on the older ones. It is then possible to generate a list of diverse concepts in a single trajectory. Second, the method generates approximately one concept per problem on average (Table 1), which defeats the purpose of exploring multiple solution ideas. This low concept count is consistent with the original work, which observes that Qwen2.5 models generate noticeably

fewer concepts than their Llama-3.2 counterparts and reports only 1.13 distinct concepts per problem on HumanEval (Handa et al., 2026). The disappearance of gains is not merely an artifact of this low concept count. Llama-3.2-3B-Instruct generates 3.18 concepts per problem on MATH500, 3 times more than the Qwen models, yet under exploratory sampling concept guidance still does not help: it reaches 86.7% pass@50 versus 88.1% for the repeated-sampling baseline (a -1.4 point change). Even when the concept generator produces multiple concepts, allowing the baseline to explore erases the reported advantage, indicating that the baseline sampling parameters, not the number of concepts, drive the effect.

These observations motivate our concept-guided sampling procedure in Section 4.

4 Concept-Guided Sampling

We first modify concept-guided sampling as an inference-time procedure. The goal is not to train a model yet, but to test whether a better concept-generation protocol can improve exploration beyond naive repeated sampling on problems where the answer generator struggles.

Exploratory Answer Generation. We keep the exploratory decoding parameters introduced in Section 3 (temperature 1.0, top-$p = 0.95$, and top-$k = -1$) for concept-conditioned generation and repeated sampling. This keeps the comparison conservative: any improvement from concepts must exceed what a strong, exploration-enabled repeated-sampling baseline already achieves.

Single-Trajectory Concept Generation. Instead of sampling concepts one by one, we prompt the concept generator to analyze the problem and emit all useful concepts in one trajectory. Each concept is required to be high-signal, problem-specific, and non-duplicative. We parse at most ten concepts from the trajectory and condition answer rollouts on those concepts. The full concept-generator and answer-generator prompts are given in Appendices B.1 and B.2.

Focusing on Hard Problems. On easy problems, repeated sampling already solves most examples, leaving little room for concept-guided exploration to help. We therefore construct model specific hard subsets from the MATH test set: for each answer-generator size, we retain only problems for which the repeated-sampling baseline obtains 0% accuracy with 100 rollouts. Each answer generator is thus evaluated on its own filtered subset. The hard subsets shrink as the answer generator grows, from 493 problems for the 1.5B model to 178 for the 32B model, since more capable models leave fewer completely unsolved problems.

Table 2: Our concept-guided sampling (without training), on model-specific hard MATH subsets (Qwen2.5-Instruct): baseline pass@50 and concept-guided differences across CG/AG size pairs. The final row reports the size of each answer generator's hard subset.

	AG					Avg Concepts
	1.5B	3B	7B	14B	32B	
Baseline pass@50	18.4	16.6	11.0	12.6	11.8	-
CG / AG	**Difference (Concept - Baseline) pass@50**					
1.5B	-1.4	-1.0	+1.7	+6.0	+3.3	4.18
3B	+0.1	+0.4	+8.3	+4.4	+8.7	8.74
7B	+1.7	+2.1	+8.0	+3.5	+8.8	9.75
14B	-0.2	+5.2	+7.2	+7.4	+8.2	9.95
32B	+3.3	+3.8	+9.7	+7.4	+6.6	9.95
Dataset size	493	274	185	176	178	-

Inference time results. In contrast to the reproduction in Section 3, where concept guidance never helped once the baseline was allowed to explore, our procedure produces consistent gains on the hard subsets (Table 2). Almost every concept-generator (CG) / answer-generator (AG) pair improves over the repeated-sampling baseline, with gains of up to $+9.7$ pass@50 points. The improvements scale with the strength of the concept generator: the 1.5B generator helps only marginally, whereas generators of 7B and larger improve nearly every answer generator by several points. Running the iterative

protocol (Handa et al., 2026) on these same hard subsets with the same AG sampling parameters also yields gains, but smaller ones (at most $+6.8$ points, with several regressions), consistent with its far lower concept count (Appendix D.5).

Most importantly, a small concept generator can improve a much larger answer generator. The 7B generator raises the 32B answer generator by $+8.8$ points despite being far smaller than the model it steers, indicating that useful exploration can be produced with little compute and injected into a stronger, possibly frozen or closed-source, answer generator. This observation directly motivates the training setup in Section 5, where a small concept generator is trained for a larger, fixed answer generator. An example of generated concepts with our approach is given in Example 1.

These gains coincide with a large increase in the number of concepts. Whereas the original procedure produced about one concept per problem (Table 1), single-trajectory generation yields between 4 and roughly 10 concepts on average, with the stronger generators saturating the cap of ten.

Example 1: A MATH problem and example concepts

Problem:
A bag contains two red beads and two green beads. You reach into the bag and pull out a bead, replacing it with a red bead regardless of the color you pulled out. What is the probability that all beads in the bag are red after three such replacements? Express your answer as a common fraction.

Example concepts:

- Use a tree diagram to map all possible outcomes of the draws and replacements.
- Calculate the probability for each state transition step-by-step.
- Analyze the problem using a Markov chain to model the transitions between states.

Cross-family check. Our reported gains are not specific to the Qwen family. On a hard subset of 424 MATH problems that Llama-3.2-3B-Instruct fails to solve with 100 rollouts, our single-trajectory procedure (using Llama-3.2-3B as both concept and answer generator) yields a modest but positive improvement, raising pass@50 from 25.2% to 26.1% ($+0.9$ points), while producing 9.23 concepts per problem on average, far more than the one to three produced by the original iterative procedure of Handa et al. (2026); see Section 3. Notably, that original procedure *degraded* this same model's exploratory baseline; switching to single-trajectory generation both increases concept diversity and flips the effect positive, confirming the improvement is not an artifact of the Qwen family.

5 Training Concept Generators

The inference time results raise a natural question: can we *train* a small model to generate better concepts for a larger answer generator? This is useful because the larger answer generator may be impractical to train directly or its weights unavailable. In particular, this provides a "training handle" for otherwise untrainable models, including those only accessible via API. Throughout, the answer generator remains frozen and only the concept generator is trained. Figure 1 summarizes the full training loop, and Appendix A gives the formal setting with a fuller discussion of the reward design.

Concept sampling and answer rollouts. Let x denote an input problem and π_θ the concept generator (CG); the answer generator (AG) is a fixed model used only for inference. For each problem, π_θ samples G trajectories. Each trajectory contains a problem analysis followed by a list of concepts. We parse the generated text and keep at most 10 concepts. For each concept trajectory, we assign a total answer-rollout allocation of B calls to the answer generator (AG). The allocation is distributed as evenly as possible across the parsed concepts: if M concepts are available, each concept receives either $\lfloor B/M \rfloor$ or $\lceil B/M \rceil$ rollouts, with the total number of rollouts equal to B. This handles cases such as $B = 128$ and $M = 10$, where some concepts receive 13 rollouts and the rest receive 12.

Reward aggregation. Each AG rollout is graded for correctness against the ground-truth answer using an LLM judge. We use an LLM judge because, even after extracting the final answer, rule-based matching can penalize mathematically equivalent answers expressed in different forms or formats. We then aggregate the rollout scores into a scalar reward for the entire concept trajectory. Consider

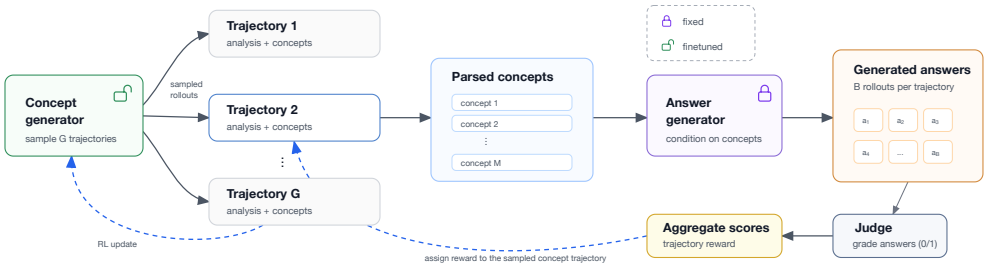


Figure 1: Overview of concept-generator training. The concept generator samples multiple concept trajectories. For one sampled trajectory, parsed concepts condition a fixed answer generator, which produces answer rollouts. A judge grades the answers, and the scores are aggregated into a trajectory-level reward used to finetune the concept generator.

a trajectory τ with parsed concepts $c_1, \ldots, c_M$, where concept c_j receives n_j answer rollouts with binary correctness $o_{j,\ell} \in \{0, 1\}$. We consider two aggregation functions,

$$r_{\text{max-max}}(\tau) = \max_j \max_\ell \; o_{j,\ell}, \qquad r_{\text{max-mean}}(\tau) = \max_j \frac{1}{n_j} \sum_{\ell=1}^{n_j} o_{j,\ell}. \tag{1}$$

Max-of-max assigns reward 1 if any answer rollout generated from any concept in the trajectory is correct, and 0 otherwise; it rewards discovering at least one concept that unlocks a correct solution. **Max-of-mean** takes the maximum over concepts of the per-concept accuracy; it rewards concepts that are reliably useful rather than concepts that only occasionally lead to a correct sample. The two aggregations therefore differ in range: max-of-max yields a binary reward in $\{0, 1\}$, whereas max-of-mean yields a continuous reward in $[0, 1]$.

Trajectory-level credit. To encourage a diverse set of concepts, we assign the aggregate reward to the entire concept trajectory instead of attributing reward to individual concepts. This encourages exploration over sets of concepts, but it may introduce position effects or allow weak concepts to share credit with strong ones. We study these effects in Appendix D.2.

Optimization. We train the CG policy π_θ with a GRPO-style objective (Shao et al., 2024). The full objective and optimization details are provided in Section 6 and Appendix C.

6 Experiments

6.1 Setup

We now test our approach introduced in the previous section to train the CG model.

Models. We use Qwen2.5-7B-Instruct as the trainable concept generator and Qwen2.5-32B-Instruct as the primary answer generator (Yang et al., 2024). The answer generator is kept fixed and used solely for inference. The transfer experiments in Section 7 further use Llama-3.3-70B-Instruct (Grattafiori et al., 2024) as an answer generator.

Data. Training uses problems and ground truths from DeepMath-103k (He et al., 2026). We filter the training set using the answer generator and retain problems where the AG achieves less than 5% success with 128 rollouts. As the held-out evaluation set we reserve 1k problems, disjoint from training, where the AG achieves 0% success with 128 rollouts; these are the hardest problems for the answer generator and are used for all main results. Because this filter is a finite-sample estimate, a fresh independent draw can still solve a fraction of these problems; we quantify this effect in Appendix D.6. To test generalization beyond the training distribution, we additionally evaluate on Omni-MATH 2 (Ballon et al., 2026), itself a filtered version of Omni-MATH (Gao et al., 2025), which we filter with the exact same procedure (retaining problems where the AG achieves 0% success with 128 rollouts) to obtain an out-of-distribution hard set. We do not train on this dataset; it is used only for evaluation, following the same protocol as the DeepMath held-out set.

Training Algorithm. We train the CG with GRPO-style reinforcement learning, using the variant adopted in OLMo3 (Olmo et al., 2025). For each problem, the CG samples $G = 8$ concept trajectories. Each trajectory receives a downstream reward computed from $B = 128$ AG rollouts. Because computing each trajectory reward requires many AG rollouts, we do not use dynamic/active sampling during training. We train for 200 batches and report both reward-aggregation variants defined in Section 5 (max-of-max and max-of-mean). Detailed hyperparameters are reported in Appendix C.

Answer Generation and Judging. The AG uses temperature 1.0, top-$p = 0.95$, top-$k = -1$, a maximum response length of 4096 tokens, and a maximum prompt length of 2048 tokens. We judge answer correctness with Qwen2.5-32B-Instruct using greedy decoding and a binary verifier prompt. We deliberately reuse the answer-generator model as the judge for efficiency: a separate high-quality judge would need to be comparably large and would consume dedicated GPUs, slowing an already compute-heavy training loop. The judge is invoked in a separate scoring prompt that compares a candidate answer after extraction to the ground truth. The judge prompt is given in Appendix B.5.

Baselines. We compare:

- **Naive repeated sampling:** sample answers directly from the AG with the same rollout allocation.
- **Generic prompt modification:** sample answers using the concept conditioned AG prompt, using ten fixed, problem-agnostic hints (e.g. "break the problem into smaller subproblems"), with the rollout allocation split across them exactly as in concept guided sampling. The full list of hints is given in Appendix B.4.
- **Untuned concept generation (Ours untrained):** use pretrained concept generators, including larger models, without RL training, as in the inference-time procedure of Section 4.
- **Trained concept generation (Ours trained):** the RL-trained concept generator steering the fixed answer generator.

Metric. The primary metric is pass@k as a function of the answer-rollout allocation k, computed with the standard unbiased estimator of Chen et al. (2021). Unless otherwise stated, curves are evaluated on the 1k DeepMath held-out set of hard problems.

6.2 Results

Table 3: Pass@k (%) on DeepMath held-out hard set and Omni-MATH 2 OOD hard set. AG is Qwen2.5-32B-Instruct, CG is Qwen2.5-7B-Instruct. Best per column in **bold**, second best in *italic*.

	DeepMath (held-out hard)			Omni-MATH 2 (OOD)		
Method	pass@1	pass@64	pass@128	pass@1	pass@64	pass@128
Baselines						
Naive repeated sampling	0.23	11.35	19.00	0.14	6.81	11.29
Generic prompt modification	0.32	14.54	22.60	0.15	6.93	11.22
Untuned concept generation (ours, no RL training)						
Untuned CG (32B)	0.82	23.77	33.80	0.27	*10.18*	*15.27*
Untuned CG (7B)	0.77	20.62	28.90	*0.28*	9.64	14.98
Trained concept generation (ours, 7B CG)						
max-of-max	*1.48*	*26.66*	*35.30*	0.27	10.04	*15.27*
max-of-mean	**2.18**	**29.64**	**39.20**	**0.44**	**12.75**	**18.60**

Main Results. The trained concept generators substantially improve pass@k over naive repeated sampling on the held-out hard problems (Table 3). The best variant (max-of-mean) reaches 39.2% pass@128, roughly double the 19.0% of naive repeated sampling, and the gap holds at smaller allocations (29.6% vs. 11.4% at pass@64). A generic, problem-agnostic prompt modification helps only modestly (22.6% pass@128), far below problem-specific concepts. Both aggregation objectives improve over the baselines, with max-of-mean ahead of max-of-max at every allocation (39.2% vs. 35.3% pass@128). This indicates the CG learns to propose problem-specific directions that reshape the AG's exploration distribution rather than merely drawing more samples from the same

distribution. We verify this exploration effect directly: embedding the generated chains of thought and measuring their diversity with the Vendi Score shows that concept guidance, and especially the trained CG, produces more diverse reasoning than naive sampling (Appendix D.8). The improvement carries over to the out-of-distribution Omni-MATH 2 set, where max-of-mean is again the best method (18.6% vs. 11.3% pass@128 for naive), though margins are smaller and max-of-max there only matches the untrained concept generators. The main table reports the headline allocations $k \in \{1, 64, 128\}$; the full pass@k curves for all methods and both datasets, together with 95% bootstrap confidence intervals and paired-bootstrap tests of the gains over naive sampling, are reported in Appendix D.7. We train for 200 steps throughout; Appendix D.9 shows that training substantially longer yields no sustained gain: performance saturates after a few hundred steps, and the max-of-mean objective eventually becomes unstable, which justifies this choice.

Small Concept Generators Can Help Larger Answer Generators. A central finding is that a small *trained* CG outperforms a much larger *untrained* one. Without training, a larger concept generator does help more, consistent with scale: the untuned 32B CG reaches 33.8% pass@128 versus 28.9% for the untuned 7B CG. After RL training, however, the 7B CG surpasses the untuned 32B CG (39.2% vs. 33.8% pass@128 for max-of-mean, and 35.3% for max-of-max), despite steering an answer generator more than four times its size. This supports the view that the concept generator learns a specialized search policy rather than relying on model scale, and that useful exploration can be produced with little compute and injected into a larger frozen answer generator. A paired bootstrap over held-out problems confirms this gap: +5.4 points pass@128 on DeepMath and +3.3 on Omni-MATH 2, with both 95% intervals excluding 0 (Table 9 in Appendix D.7).

Comparison to Inference-Time Concept Sampling. The untuned concept generators in Table 3 are exactly the inference-time procedure of Section 4, and they already improve over naive sampling (28.9% pass@128 for the 7B CG). RL training adds a further +10.3 points over the untuned CG, showing that training improves concept quality. Appendix D.10 shows qualitative examples of how training reshapes generated concepts into more concrete, problem-specific plans. We confirm this improvement is statistically significant with a paired bootstrap over held-out problems (per-problem difference of the two unbiased pass@k estimates); Table 10 in Appendix D.7 reports the trained over untuned 7B gain for both reward objectives and datasets.

7 Analysis and Ablations

7.1 Do Problem-Relevant Concepts Matter?

We test whether the gains come from problem-specific guidance or merely from perturbing the prompt. Using the trained CG concepts (max-of-mean), we apply a random derangement over the held-out set so that each problem is paired with another problem's concept trajectory (no problem keeps its own), and re-run answer generation. Mismatched concepts still help slightly over naive sampling (23.9% vs. 19.0% pass@128), on par with the generic prompt-modification baseline (22.6% pass@128), consistent with a concept from an unrelated problem acting as a generic perturbation. But they fall far short of matched concepts (39.2% pass@128): problem relevance accounts for roughly 15 of the 20 point improvement over naive sampling. This indicates that most of the benefit comes from problem-specific concepts, not from generic prompt perturbation.

7.2 Transfer Across Datasets and Models

Cross-model transfer. Our concept generator is trained only against the Qwen2.5-32B-Instruct answer generator and never sees Llama-3.3-70B-Instruct (Grattafiori et al., 2024) during training. To test whether the learned search policy transfers, we take the trained 7B CG and use it to steer Llama-3.3-70B, a larger model from a different family, on both the 1k DeepMath held-out set and the Omni-MATH 2 set. We reuse both filtered sets exactly as in Section 6 (filtered so the Qwen2.5-32B AG scores 0%, and reused as-is rather than re-filtered for Llama), so they measure transfer on common hard sets rather than Llama's own worst-case problems. Accordingly the Llama baseline is non-zero, solving 6.2% of the DeepMath problems at pass@1. For all rows we judge correctness with Qwen2.5-32B-Instruct, consistent with our earlier experiments. As shown in Table 4, on DeepMath the transferred CG is best at every k value, reaching 34.3% pass@128 versus 26.2% for naive sampling and 28.9% for Llama generating its *own* concepts. The small trained CG thus outperforms

Table 4: Transfer to Llama-3.3-70B as the answer generator, on the DeepMath and Omni-MATH 2 hard sets. Pass@k (%); best per column in **bold**, second best in *italic*.

	DeepMath (held-out hard)			Omni-MATH 2 (OOD)		
Method (AG = Llama-3.3-70B)	pass@1	pass@64	pass@128	pass@1	pass@64	pass@128
Naive repeated sampling	*6.17*	22.38	26.20	**1.92**	9.63	12.50
Untuned concept generation (no RL training)						
Llama-70B self-concepts	6.15	24.91	28.90	*1.88*	9.77	11.94
Untuned 7B CG (Qwen)	5.66	*26.24*	*31.30*	1.73	*9.92*	*12.59*
Trained concept generation (7B CG, max-of-mean)						
7B CG trained vs. 32B (transfer)	**6.83**	**29.01**	**34.30**	1.73	**10.75**	**13.97**

the $10\times$larger answer generator's self-generated concepts, despite never being trained against it. This indicates the CG learns a transferable exploration policy rather than one tied to a specific AG.

7.3 Training and Inference Compute

The training loop of our approach (Section 5) is compute-heavy, and this compute is almost entirely dominated by reward estimation. Before a single gradient is taken, scoring the $G = 8$ concept trajectories for a problem requires $G \times B = 1024$ generations from the frozen answer generator plus 1024 judge calls per problem in the batch. In a representative training step of about 413 s, the AG rollouts take ~ 300 s and judging ~ 85 s (together roughly 94% of the step) while generating the concept trajectories with the CG takes only ~ 18 s and the actor update about 10 s. Training is therefore bottlenecked by repeatedly running the large model we do *not* train, not by the small concept generator we optimize. Crucially, training is run only once.

At inference, running the CG to sample one concept trajectory per problem adds negligible compute next to running B AG rollouts per problem. Using the standard estimate of $2N$ (Kaplan et al., 2020; Snell et al., 2024) FLOPs per token for a dense transformer with N parameters, a single generation requires about $2N(\ell_{\text{in}} + \ell_{\text{out}})$ FLOPs, so the compute of one CG call relative to one AG rollout is roughly $\frac{N_{\text{CG}}}{N_{\text{AG}}} \cdot \frac{\ell_{\text{CG}}}{\ell_{\text{AG}}}$. With $N_{\text{CG}} = 7.6$B and $N_{\text{AG}} = 32.5$B parameters and average lengths of $\ell_{\text{CG}} \approx 280 + 800 = 1080$ and $\ell_{\text{AG}} \approx 240 + 820 = 1060$ tokens, this ratio is about $\frac{7.6}{32.5} \cdot \frac{1080}{1060} \approx 0.24$: generating the concepts takes less than a quarter of a single answer rollout. Furthermore, it is under 0.3% of the $B = 128$ rollout answer allocation (we ignore ℓ_{in} for the AG due to prefix sharing). Therefore this allocation is essentially the *same* as naive repeated sampling. We do not add answer rollouts, we only prepend one lightweight small-model call and change how the existing B rollouts are prompted, so concept-guided exploration runs at essentially the same inference compute as the baseline.

8 Conclusion

We revisited LLM exploration beyond naive repeated sampling. After showing that prior concept-guided gains largely disappear against an exploratory baseline, we introduced a stronger procedure that generates multiple detailed concepts in one trajectory and trains a small concept generator using the downstream success of a frozen answer generator. On hard mathematical problems, the learned policy roughly doubles pass@k at the same answer-rollout allocation, outperforms concepts from much larger untuned models, and transfers to an answer generator from a different model family. These results show that a small model can serve as an effective, reusable search policy for a larger model that may be impractical to train directly or closed-source.

Several limitations remain. Training is compute heavy because reward estimation requires many answer-generator rollouts, rewards are assigned at the trajectory level rather than to individual concepts, and our evaluation focuses on mathematical reasoning. Future work should develop lower-compute reward estimation, more granular credit assignment, judge-robust evaluation, and bandit based allocation of rollouts across concepts, while extending the approach to broader domains.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing and debugging research code used to implement our method. We did not use generative AI tools to generate synthetic datasets; develop theoretical models or conceptual frameworks; formulate mathematical claims or proofs; propose or refine hypotheses; design the research methodology or experiments; clean or reformat datasets; or conduct qualitative or thematic data analysis. Additionally, we used generative AI tools to reformulate and edit manuscript text, assist with figure preparation, and find related work. We reviewed all AI-assisted work: research code was tested, experimental outputs were checked, and suggested text, figures were manually inspected by the authors. We take responsibility for the final content of this work, including text, claims, results, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work evaluates methods for improving the reasoning capabilities of large language models. The experiments use existing mathematical reasoning benchmarks and do not involve human subjects or newly collected personal data. As with other methods that improve model capabilities, any deployment should be accompanied by application specific safety evaluation and appropriate safeguards.

REPRODUCIBILITY STATEMENT

The main text specifies the proposed method, training objectives, baselines, evaluation protocol, and statistical comparisons. The appendices provide the formal analysis, full prompts, training and generation hyperparameters, dataset filtering and concept-parsing details, and additional results, needed to reproduce the reported experiments.

REFERENCES

Marthe Ballon, Andres Algaba, Brecht Verbeken, and Vincent Ginis. Benchmarks saturate when the model gets smarter than the judge. *arXiv preprint arXiv:2601.19532*, 2026.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Dan Friedman and Adji Bousso Dieng. The vendi score: A diversity evaluation metric for machine learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=g97OHbQyk1.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Zhengyang Tang, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, 2025.

Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daehoon Gwak, Minseo Jung, Junwoo Park, Minho Park, ChaeHun Park, Junha Hyung, and Jaegul Choo. Not the example, but the process: How self-generated examples enhance llm reasoning. *arXiv preprint arXiv:2602.15863*, 2026.

Divij Handa, Mihir Parmar, Aswin RRV, Md Nayem Uddin, Hamid Palangi, and Chitta Baral. Guidedsampling: Steering llms towards diverse candidate solutions at inference-time. In *International Conference on Learning Representations*, 2026.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. In *International Conference on Learning Representations*, 2026.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. Guiding large language models via directional stimulus prompting. *Advances in Neural Information Processing Systems*, 36:62630–62656, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison 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.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. Rlad: Training llms to discover abstractions for solving reasoning problems. *arXiv preprint arXiv:2510.02263*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Shobhita Sundaram, John Quan, Ariel Kwiatkowski, Kartik Ahuja, Yann Ollivier, and Julia Kempe. Teaching models to teach themselves: Reasoning at the edge of learnability. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves llm search for code generation. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. In *International Conference on Learning Representations*, 2024.

A Formal Setting and Analysis of the Reward Objectives

Objective. Our goal is exploration: finding at least one valid solution path within a sampling allocation k. This is exactly the definition of pass@k, which we report per problem x and average over the dataset.

Let $q(\cdot \mid x)$ denote the *frozen* model that generates candidate answers, and let $\pi_\theta(\cdot \mid x)$ denote the *trained* policy, which generates concepts. Write

$$p(x) \;=\; \mathbb{E}_{y\sim q(\cdot|x)}\big[\mathbb{1}\{y \text{ correct}\}\big]$$

for the probability that naive sampling from the frozen model produces a correct answer. Throughout we assume (i) rollouts are drawn i.i.d. from the relevant conditional of q, and (ii) the verifier is deterministic. Under these assumptions the baseline is

$$\text{pass@}k \;=\; 1-\big(1-p(x)\big)^k \tag{2}$$

Concept-conditioned sampling. We instead condition generation on concepts. Given a set of M concepts $c=(c_1,\dots,c_M)$ sampled from $\pi_\theta(\cdot \mid x)$, define $p(x,c_j)=\mathbb{E}_{y\sim q(\cdot|x,c_j)}[\mathbb{1}\{y \text{ correct}\}]$ and allocate $n=k/M$ rollouts to each concept (for the analysis we assume $M \mid k$, so that n is an integer, the implementation splits the allocation as evenly as possible, Section 5). We additionally assume (iii) rollouts are independent *across* concepts given c. Note that the concepts themselves are *not* independent as they are produced by a single autoregressive trajectory of π_θ, but this does not affect the product below, which is taken over rollout outcomes conditionally on c. The probability of obtaining at least one correct trajectory is then

$$\text{pass}^c\text{@}k \;=\; 1-\prod_{j=1}^{M}\big(1-p(x,c_j)\big)^n \;=\; 1-\big(1-p_{\text{eff}}(x,c)\big)^k, \tag{3}$$

where we define the *effective solve rate*

$$p_{\text{eff}}(x,c) \;=\; 1-\Big[\textstyle\prod_{j=1}^{M}\big(1-p(x,c_j)\big)\Big]^{1/M}. \tag{4}$$

Equation (3) puts concept-conditioned sampling on exactly the same footing as (2): $1-p_{\text{eff}}$ is the *geometric mean of the per-concept failure probabilities*. Moreover, p_{eff} does not depend on k, so the comparison holds for any k.

When does conditioning help? Define the yield of a concept as $\lambda(x,c_j)=-\log\big(1-p(x,c_j)\big)$, and let $\lambda_0 \;=\; -\log\big(1-p(x)\big)$ denote the baseline yield. Since $-\log\big(1-p_{\text{eff}}(x,c)\big) \;=\; \frac{1}{M}\sum_{j=1}^{M}\lambda(x,c_j) \;=:\; \bar\lambda(x,c)$, equations (2) and (3) read pass@$k \;=\; 1-e^{-k\lambda_0}$ and pass^c@$k \;=\; 1-e^{-k\bar\lambda(x,c)}$, so that

$$\text{pass}^c\text{@}k \;>\; \text{pass@}k \qquad\Longleftrightarrow\qquad \bar\lambda(x,c) \;>\; \lambda_0. \tag{5}$$

Conditioning helps precisely when it raises the average per-concept yield above the baseline. This also means that every concept must earn its share of the allocation. Adding a further concept c_{M+1} replaces $\bar\lambda$ by $\big(M\bar\lambda+\lambda(x,c_{M+1})\big)/(M+1)$, which improves pass^c@$k$ if and only if

$$\lambda(x,c_{M+1}) \;>\; \bar\lambda(x,c). \tag{6}$$

A concept that is merely below the current average is not neutral but actively harmful: it consumes k/M rollouts while contributing less yield. This dilution effect is the central drawback of concept-conditioned sampling, and it is what any useful reward must penalise. We note in passing that, because λ is convex in p, Jensen's inequality gives $p_{\text{eff}}(x,c) \;\geq\; \bar p(x,c) \;=\; \frac{1}{M}\sum_j p(x,c_j)$, with equality iff all $p(x,c_j)$ coincide, so $\bar p(x,c) > p(x)$ is sufficient for improvement, though (5) is the exact criterion.

Why we evaluate on problems where the baseline is weak. The headroom available to any method is $1-\text{pass@}k \;=\; (1-p(x))^k$, which decays exponentially in $p(x)$. At $p(x) \;=\; 0.1$ and $k=128$ the baseline already reaches pass@$k \approx 1-1.4\times10^{-6}$, leaving essentially nothing to win. We therefore focus on problems for which $p(x)$ is small, which is exactly what our evaluation filter selects. Beyond this ceiling effect, there is a second reason to expect no gain when $p(x)$ is already high: conditioning on a concept restricts q to a narrower region of the solution space, so when the unconditional model already covers that space well, steering it toward particular exploration directions would offer little upside.

Rollouts allocation. Our uniform allocation $n_j = k/M$ is not optimal. Maximising $1 - \prod_j (1 - p(x, c_j))^{n_j}$ subject to $\sum_j n_j = k$ is equivalent to maximising the linear objective $\sum_j n_j \lambda(x, c_j)$, whose solution is degenerate: an oracle with knowledge of the $p(x, c_j)$ would assign the entire allocation to the single best concept, achieving $1 - (1 - p_{\max}(x, c))^k$ with $p_{\max}(x, c) = \max_j p(x, c_j)$. This gives the chain

$$\bar{p}(x, c) \;\le\; p_{\text{eff}}(x, c) \;\le\; p_{\max}(x, c), \tag{7}$$

whose three terms correspond respectively to mean concept quality, to the effective quality realised under uniform allocation, and to the quality realised under oracle allocation. Closing the second gap is naturally posed as a *bandit problem* over concepts. We retain uniform allocation for throughput reasons, and leave adaptive allocation which can give better performance to future work.

Reward design. The *max-of-max* reward is unbiased for the objective we actually deploy: conditioned on the generated concepts c, we have $r_{\text{max-max}} \mid c \sim \text{Bernoulli}\big(\text{pass}^c@k\big)$, and in particular $\mathbb{E}\big[r_{\text{max-max}} \mid c\big] = \text{pass}^c@k$. It is, however, a one-bit measurement: it collapses all k rollout outcomes into a single indicator, leaving almost no basis for comparing concept sets within a group. In particular, whenever every set in a group succeeds or every set fails, the group-normalised advantages are identically zero and the update contributes no gradient which corresponds to the vanishing advantage regime documented for binary verifiable rewards (Yu et al., 2026).

We therefore also consider the *max-of-mean* reward, which averages the k/M rollouts within each concept before aggregating across concepts, retaining $O(\log(k/M))$ bits rather than one. It rewards generating at least one concept with a high solve rate $p(x, c_j)$, and is a monotone estimate of $p_{\max}(x, c)$. By (7), this means $r_{\text{max-mean}}$ is order preserving for the pass$^c@k$ attainable under *oracle* allocation rather than under the uniform allocation we deploy: its bias is precisely the allocation gap $p_{\max} - p_{\text{eff}}$, and it would become the aligned reward under the adaptive allocation discussed above. Two consequences of this mismatch are worth noting: by replacing the noisy-OR over concepts with a hard maximum, $r_{\text{max-mean}}$ does not credit sets that succeed through the combination of several moderate concepts and it inherits the upward bias of a maximum over noisy estimates ($\mathbb{E}\left[r_{\text{max-mean}} \mid x, c\right] \ge p_{\max}(x, c)$). We nonetheless adopt it because in our regime the resulting gain in reward informativeness outweighs this bias, as we show in Section 6.

Choice of M. The number of concepts is bounded by two competing effects. Additional concepts widen the search over exploration directions, but by (6) a concept improves pass$^c@k$ only if its yield exceeds the current average: a condition that marginal concepts increasingly fail as M grows, so that dilution eventually dominates. At the same time, larger M reduces the rollout count $n = k/M$ per concept, making each $\hat{p}(x, c_j)$ noisier and strengthening the upward bias of $r_{\text{max-mean}}$.

Estimating pass$^c@k$ from a larger rollout pool. In evaluation we generate B rollouts and use the full pool to estimate performance at smaller allocations $k \le B$, rather than discard $B - k$ outcomes and evaluate a single subset. As in Chen et al. (2021), averaging over all admissible k-subsets yields a lower-variance estimate of the corresponding pass probability. For clarity, suppose that M divides both B and k, let $b = B/M$ and $n = k/M$, and let S_j be the number of successes among the b rollouts assigned to concept c_j. An exactly unbiased estimator for the fixed balanced allocation is

$$\widehat{\text{pass}^c@k}_{\text{strat}} = 1 - \prod_{j=1}^{M} \frac{\binom{b-S_j}{n}}{\binom{b}{n}}. \tag{8}$$

Indeed, $\frac{\binom{b-S_j}{n}}{\binom{b}{n}}$ is an unbiased estimator, based on b observed rollouts, of the probability that n rollouts conditioned on concept c_j all fail. Since answer rollouts are independent conditional on $c_1, \ldots, c_M$, multiplying these per-concept failure probability estimators and subtracting the result from one, gives the estimator in (8).

In our results we instead use the standard pooled estimator of Chen et al. (2021), $1 - \binom{B-S}{k} / \binom{B}{k}$, where $S = \sum_j S_j$, for simplicity and direct comparability with the naive baseline. This estimator is not exactly unbiased for a fixed balanced allocation when the $p(x, c_j)$ differ: it is unbiased for selecting a uniformly random k-subset of the pooled B rollouts, which induces random per-concept

counts $A = (A_1, \ldots, A_M)$ with a multivariate hypergeometric distribution and $\mathbb{E}[A_j] = n$. Writing $\lambda_j = \lambda(x, c_j) = -\log(1 - p(x, c_j))$, Jensen's inequality gives

$$\mathbb{E}\left[\widehat{\text{pass@}k}_{\text{Chen}} \mid c\right] = 1 - \mathbb{E}_A\left[e^{-\sum_j A_j \lambda_j}\right] \leq 1 - e^{-n\sum_j \lambda_j} = \text{pass}^c@k, \tag{9}$$

so the pooled estimator is conservative in the uniform allocation setting. Moreover, if $v_\lambda = M^{-1}\sum_j(\lambda_j - \bar{\lambda})^2$, using a second order approximation gives the following leading discrepancy:

$$\text{pass}^c@k - \mathbb{E}\left[\widehat{\text{pass@}k}_{\text{Chen}} \mid c\right] \simeq \frac{1}{2}\big(1 - \text{pass}^c@k\big)\frac{k(B-k)}{B-1}v_\lambda, \tag{10}$$

which is second order in the variation of the per-concept yields. Their expectations coincide when all concepts have the same success probability, and the estimators themselves coincide exactly at $k = B$. Hence our headline pass@128 values with $B = 128$ are unaffected. Moreover, in our low success regime the per-concept solve rates are small (Appendix D.2), so the correction in (10) is correspondingly small. We therefore retain the pooled Chen estimator as a simple, common estimator for all methods and allocations.

B PROMPTS

This appendix records the prompts used for concept generation, answer generation, baseline answer generation, and answer judging.

B.1 CONCEPT GENERATOR PROMPT

The concept generator is prompted to first analyze the problem and then enumerate a short list of high-signal, non-duplicated concepts, theorems, and creative problem-solving techniques, while being explicitly instructed not to solve the problem or reveal the final answer. The system and user messages are shown below.

Concept Generator Prompt

```
System:
You are an expert mathematician.
You will be given a mathematical problem. First, carefully
analyze the problem:
identify what type of problem it is, what mathematical
structures are present, and what approaches might work.
Then, list ALL relevant mathematical theorems, fundamental
concepts, hints, and creative problem-solving techniques
that could be useful for solving the problem.
Do NOT solve the problem. Do NOT compute the final answer.

OUTPUT FORMAT (MUST FOLLOW EXACTLY):
First, write your analysis of the problem.
Then, list concepts. Each concept line MUST start with
exactly: ####
After '#### ' write ONE concept/theorem or a sentence for
the creative technique, for example: #### your concept or
technique here.
Each concept must be high-signal and specific enough (avoid
generic items like 'algebra' or 'logical reasoning').
List between 1 and 10 concepts.
NO DUPLICATES OR NEAR DUPLICATES. The techniques or concepts
should be new and different than previously generated ones.
Don't write anything after the end of the last concept.

User:
```

```
QUESTION:
{problem}
```

B.2 Concept-Conditioned Answer Generator Prompt

The answer generator is prompted to produce a step-by-step solution to the problem while conditioning on a single concept, which is injected as a hint in the user message. The system and user messages are shown below.

Concept-Conditioned Answer Generator Prompt

```
System:
You are an expert mathematician. Your task is to answer
mathematical questions
with step-by-step solution.
Follow these instructions precisely:
Present your solution as a step-by-step process.
Explain each step clearly and concisely.
Use correct mathematical notation throughout your solution.
Output the final answer within \boxed{}.

User:
QUESTION:
{problem}

Here is a potentially useful hint: {concept}
Let's think step by step and output the final answer within
\boxed{}.
```

B.3 Repeated-Sampling Baseline Prompt

The repeated-sampling baseline uses the same answer-generator prompt as Appendix B.2 but without any concept hint, so that all rollouts are drawn from the unconditioned answer generator. The system and user messages are shown below.

Repeated-Sampling Baseline Prompt

```
System:
You are an expert mathematician. Your task is to answer
mathematical questions with step-by-step solution.
Follow these instructions precisely:
Present your solution as a step-by-step process.
Explain each step clearly and concisely.
Use correct mathematical notation throughout your solution.
Output the final answer within \boxed{}.

User:
QUESTION:
{problem}

Let's think step by step and output the final answer within
\boxed{}.
```

B.4 Generic Prompt Baseline

The generic prompt-modification baseline uses the exact same concept-conditioned answer-generator prompt (Appendix B.2), but replaces the problem-specific concept in the `{concept}` slot with one of ten fixed, problem-agnostic hints. The ten generic hints are:

Generic Hints

```
Try a different angle than the obvious one.
Use the classical, textbook approach to solve this.
Think outside the box and look for a creative shortcut.
Break the problem into smaller, simpler subproblems.
Look for symmetry, patterns, or invariants you can exploit.
Work backwards from the desired result to the given
information.
Consider small or special cases first to build intuition.
Translate the problem into algebra and manipulate the
equations carefully.
Reformulate the problem geometrically or visually.
Estimate or bound the answer before computing it exactly.
```

B.5 Judge Prompt

We grade answers with an LLM judge that compares the student answer against the ground-truth answer and returns a binary correctness verdict. The system and user messages are shown below.

Judge Prompt

```
System:
You are a math verifier. Compare between the student answer
and the given correct ground truth.
Reply ONLY 'correct' or 'incorrect'.

User:
problem: {problem}

student answer: {answer}

ground truth: {ground_truth}

verdict:
```

B.6 Leakage-Detection Judge Prompt

To audit for answer leakage, we use a separate LLM judge that, given the ground-truth answer and a single concept, decides whether the concept states or reveals the final answer. The system and user messages are shown below.

Leakage-Detection Judge Prompt

```
System:
You are a strict grader whose job is to detect answer
leakage.
```

```
You are given the GROUND-TRUTH final answer to a math
problem and a single
CONCEPT. A concept is a hint or exploration strategy that is
meant to guide someone toward solving the problem WITHOUT
revealing the final answer.
Decide whether the concept reveals the answer. Answer 'yes'
if the concept explicitly states the ground-truth answer, or
states a numeric value or expression that is mathematically
equivalent to the ground-truth answer.
Answer 'no' if the concept only describes methods,
strategies, intuitions, reformulations, or partial setups
that do not state the final answer value.
Do not try to solve the problem yourself; judge only whether
the answer value is present in the concept.
Reply ONLY 'yes' or 'no'.

User:
ground truth answer: {ground_truth}

concept: {concept}

Does the concept state or reveal the ground-truth answer (or
a value
mathematically equivalent to it)? Reply ONLY 'yes' or 'no'.
verdict:
```

C Training Details

This appendix gives the full training and generation configuration. The main text summarizes the setup in Section 6. We implement our training loop on top of verl (Sheng et al., 2024).

Table 5: Training and generation hyperparameters.

Component	Hyperparameter	Value
CG training	Learning rate	1×10^{-6}
CG training	Warmup steps	50
CG training	Optimizer	AdamW
CG training	AdamW betas	$(0.9, 0.99)$
CG training	KL coefficient	0.001
CG training	PPO clip low / high	0.2 / 0.272
CG training	Train batch size	64
CG training	PPO mini-batch size	32
CG training	Loss aggregation	token-mean
CG generation	Temperature	0.8
CG generation	top-p / top-k	0.95 / 100
CG generation	Max prompt / response length	2048 / 2048
RL rollout	Trajectories per problem G	8
RL rollout	AG rollout allocation per trajectory B	128
AG generation	Temperature	1.0
AG generation	top-p / top-k	0.95 / -1
AG generation	Max prompt / response length	2048 / 4096
Judge	Model	Qwen2.5-32B-Instruct
Judge	Decoding	Greedy

C.1 Training Objective

The main text (Section 5) gives the group-relative advantage $A_i = r_i - \frac{1}{G}\sum_{j=1}^{G} r_j$ used to train the concept generator π_θ. Here we state the complete GRPO-style objective, as used in Olmo3 (Olmo et al., 2025), including the clipped policy-gradient term and the KL penalty to a fixed reference policy:

$$\mathcal{J}(\theta) = \frac{1}{\sum_{i=1}^{G}|y_i|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\left[\min\left(\frac{\pi_{\theta_{\text{old}}}(y_{i,t}|x,y_{i,<t})}{\pi^{\text{vllm}}_{\theta_{\text{old}}}(y_{i,t}|x,y_{i,<t})},\rho\right)\min(w_{i,t}\,A_{i,t},\ \text{clip}(w_{i,t},\ 1-\varepsilon_{\text{low}},\ 1+\varepsilon_{\text{high}})\,A_{i,t}) - \beta\,\mathbb{D}_{\text{KL}}(\pi_\theta\,\|\,\pi_{\text{ref}})\right]$$

Here G is the group size, i.e. the number of concept trajectories sampled per problem, and y_i is the token sequence of trajectory i (with length $|y_i|$), so the objective is normalized per token (token-mean) across the group. The ratio $w_{i,t} = \pi_\theta(y_{i,t} \mid x, y_{i,<t})/\pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t})$ is the usual token-level policy ratio, and the leading $\min(\cdot,\rho)$ factor is a truncated importance-sampling correction between the rollout policy $\pi^{\text{vllm}}_{\theta_{\text{old}}}$ that generated the trajectory under vLLM and the same policy $\pi_{\theta_{\text{old}}}$ as recomputed by the training backend, capped at $\rho = 2.0$. We use an asymmetric (clip-higher) clipping range $\varepsilon_{\text{low}} = 0.2$, $\varepsilon_{\text{high}} = 0.272$, and do not normalize advantages by their standard deviation. The advantage $A_{i,t} = A_i$ is the trajectory-level group-relative advantage broadcast to every token of trajectory i, where the trajectory reward r_i is the aggregate scalar obtained from the binary $(0/1)$ correctness scores of the answer-generator rollouts for that concept trajectory (max-of-max or max-of-mean, Section 5). The KL penalty $\mathbb{D}_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$ uses the low-variance estimator $\frac{\pi_{\text{ref}}}{\pi_\theta} - \log\frac{\pi_{\text{ref}}}{\pi_\theta} - 1$ with coefficient $\beta = 0.001$; the reference policy π_{ref} is the concept generator before any RL training and is held fixed throughout.

C.2 Dataset Filtering

We filter DeepMath-103k (He et al., 2026) using the answer generator (Qwen2.5-32B-Instruct) and retain training problems where the AG obtains less than 5% success rate estimated with 128 rollouts, i.e. at most 6 correct answers out of 128. Success is judged by Qwen2.5-32B-Instruct, and the filtering rollouts use the same AG decoding parameters as in Table 5 (temperature 1.0, top-$p = 0.95$, top-$k = -1$). From the filtered pool we reserve a held-out evaluation set of 1k problems with 0% success rate estimated with 128 rollouts, disjoint from training; this leaves 10,776 problems for training. We filter at 5% success rate for training, while requiring a 0% success rate for evaluation, ensuring that the training set still provides a learnable signal.

To evaluate out-of-distribution generalization, we filter Omni-MATH-2 (Ballon et al., 2026), itself a filtered version of Omni-MATH (Gao et al., 2025), with the same procedure, keeping only problems with 0% success rate estimated with 128 rollouts. This yields 1,382 problems, which are used only for evaluation.

C.3 Concept Parsing

The CG prompt requires every concept to start with the delimiter `####`. Parsing is regex-based and lenient: any analysis or preamble before the first `####` is ignored, whitespace-only spans are dropped, and malformed text simply yields no matches rather than raising an error. We keep at most the first ten parsed concepts per trajectory (`max_concepts_per_trajectory` $= 10$); when a trajectory exceeds this, its response tokens are truncated on-policy at the cut point so that the retained tokens keep their original conditional probabilities.

Empty concept lists. A trajectory with no valid `####` marker parses to an empty list. During training such trajectories generate no answer rollouts and receive a fixed penalty reward of -0.1, which mildly discourages degenerate outputs; during validation they are replaced by a single empty concept so the pipeline still runs. We track the fraction of empty trajectories as a metric; in practice it stays at roughly 0% throughout training, so parsing works out of the box.

Duplicates. Duplicate concepts are not removed for training or answer generation: identical concepts, if they appear, are treated as distinct list positions, i.e. we don't apply any deduplication.

Length limits. If a single concept would exceed the remaining allowed prompt length (maximum prompt length minus the problem and template, with a small safety margin), it is truncated to fit.

D ADDITIONAL RESULTS

This appendix presents additional empirical results omitted from the main paper for space, including sanity checks, a direct-training comparison, full pass@k curves with confidence intervals, and analyses of concept position and count.

D.1 TRAINING THE SMALLER MODEL DIRECTLY

A second question is whether the CG+AG setup is even necessary: could we instead simply train the small model to answer the hard problems on its own and skip the answer generator altogether? This matters because in our setting the AG is frozen and possibly closed-source, so the only trainable alternative is the small model itself. To test this, we train an identical 7B model with the same RL algorithm, training split, and LLM judge, but with a direct answer objective, and compare it against our CG-steered 32B across the rollout allocation (Figure 2). At pass@128, the trained 7B CG steering the 32B reaches 39.2%, compared with 24.7% for the trained direct-answer model. Moreover, even the *untuned* 7B CG steering the 32B reaches 28.9% pass@128, already above the trained direct-answer model. **Training the small model to answer directly does not recover the performance of using it as a concept generator for a larger model: the value of our approach is injecting exploration into a stronger frozen answer generator at scale, not having the small model solve problems by itself.** The direct-answer model does reach 16.3% pass@1 (and 24.7% pass@128) on problems where the 32B AG scored 0% at 128 rollouts; this reflects the finite-sample nature of the filter (a different model recovers a fraction of these problems, Appendix D.6) rather than genuine 0% difficulty.

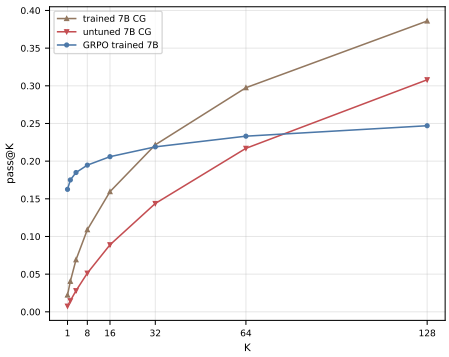


Figure 2: Held-out pass@k on the 1k hard set: a 7B trained to answer directly, our trained 7B CG steering the 32B, and the untuned 7B CG steering the 32B.

D.2 POSITION BIAS UNDER TRAJECTORY-LEVEL REWARDS

Because the reward is assigned to the whole concept trajectory rather than to individual concepts (Section 5), weak concepts could in principle free-ride on strong ones, and the model could learn to pad later positions with filler. We check for this by measuring, for each position in the trajectory, the mean accuracy of the AG rollouts conditioned on the concept at that position (the same per-concept mean used by max-of-mean), averaged over the trajectories that contain a concept at that position. Coverage stays high throughout: every position is present in at least $\sim 89\%$ of trajectories (positions receive between 11,580 and 13,084 rollouts), so the pattern is not an artifact of missing late concepts. Per-position accuracy varies only within a narrow band: it ranges from 1.3% at the first concept to 2.7% at the fifth, with all remaining positions clustered near 2%. Crucially, no position is dead: every position yields correct solutions and none dominates. The best position is only about twice the weakest, while positions 2–10 have broadly comparable estimated accuracies and overlapping bootstrap confidence intervals. The *first* concept is the weakest, suggesting that useful ideas are not concentrated at the start. This supports our use of trajectory-level credit: it does not degenerate into a few useful positions plus dead ones.

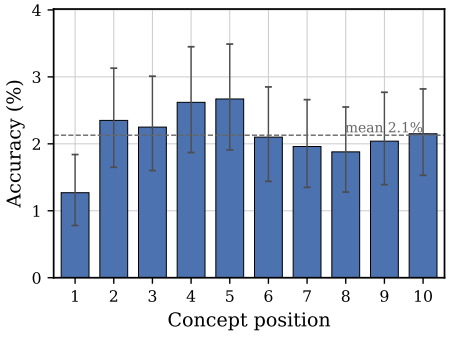


Figure 3: Per-concept accuracy by position in the trajectory (trained 7B CG, max-of-mean). Error bars are 95% bootstrap CIs over the held-out problems (10,000 resamples, 2.5/97.5 percentiles).

D.3 Answer Leakage

A natural worry is reward hacking: the concept generator could learn to solve each problem then pass the final answer into its "concepts," in which case the downstream gains would reflect answer leakage rather than better exploration. We test this directly by checking whether the ground-truth answer is present in the generated concepts. For each held-out problem we run an LLM judge (Qwen2.5-32B-Instruct) over each parsed concept and ask whether it states, or gives a value mathematically equivalent to, the ground-truth answer; the judge prompt is given in Appendix B.6. We deliberately avoid exact string matching, which produces spurious hits when the ground-truth answer is a short value that happens to coincide with unrelated text in a concept. We find that only 0.44% of concepts are flagged as containing the answer, and just 3% of problems have at least one flagged concept. This confirms that the CG steers exploration rather than leaking solutions.

D.4 Effect of the Number of Concepts

After training, the CG almost always still emits close to ten concepts (we truncate to the first ten), so the count does not collapse during training. To separate the value of using *multiple* concepts from the value of simply spending more compute, we hold the total answer allocation fixed. For each held-out problem, we first sample one generated concept uniformly at random, then incrementally sample additional concepts without replacement to form nested sets of sizes $M \in \{1, 2, 4, 8, 10\}$. Thus every larger set retains all concepts from the smaller settings. At each M, we split the same $B = 128$ rollout allocation as evenly as possible across the selected concepts (so $M = 1$ concentrates all 128 rollouts on one concept, whereas $M = 10$ assigns 12 or 13 rollouts to each). Under this nested comparison, more concepts monotonically improve performance for $k \geq 4$: pass@128 rises from 22.70% with one concept to 28.10%, 32.70%, 36.50%, and 39.80% with 2, 4, 8, and 10 concepts, respectively. pass@64 similarly rises from 16.87% to 30.34% (Figure 4). In contrast, pass@1 remains in the narrow range 2.15–2.44%. This widening benefit with k is the signature of improved *exploration* rather than single-shot accuracy: spreading a fixed allocation over additional, diverse concepts covers more of the solution space than repeatedly sampling from one. The incremental gains diminish as M grows but remain positive through ten concepts. Because total compute is fixed and the concept sets are nested, the trend cannot be explained by extra sampling or by independently redrawing a more favorable subset at each M: it directly supports our choice to generate many concepts rather than one.

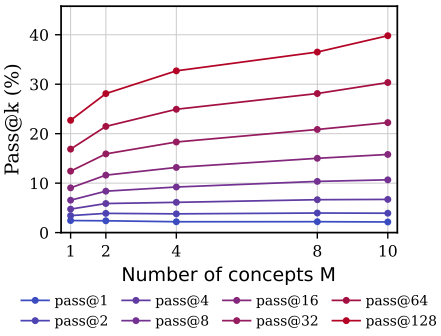


Figure 4: Pass@k vs. number of concepts M at a fixed total allocation ($B = 128$ split across the M concepts); concept sets are nested as M increases, and k ranges from 1 to 128 (blue to red). Trained 7B CG (max-of-mean) steering the 32B AG over the 1k held-out set.

D.5 Iterative Guided-Sampling Protocol on the Hard Subsets

For a like-for-like comparison with our inference-time procedure (Section 4), we run the original iterative guided-sampling protocol of Handa et al. (2026) on the *same* model-specific hard MATH subsets, using the *same* favorable exploratory decoding parameters for the answer generator (temperature 1.0, top-$p = 0.95$, top-$k = -1$). Only the concept-generation procedure differs: concepts are produced one at a time with the original iterative protocol rather than in a single trajectory. Table 6 reports the resulting pass@50 differences over the repeated-sampling baseline; the baseline pass@50 values are identical to those in Table 2.

The original protocol does yield some improvements on these hard subsets once the answer generator is allowed to explore, but they are consistently smaller than ours (Table 2): its largest gain is $+6.8$ points (vs. our $+9.7$), and several CG/AG pairs still regress. We attribute the gap to concept count: because the iterative protocol samples concepts one by one, it produces at most 1.5 concepts per

problem on average (for the 32B generator), whereas our single trajectory procedure produces up to ~ 10.

Table 6: Iterative guided-sampling protocol (Handa et al., 2026) (without training) on the same model specific hard MATH subsets (Qwen2.5-Instruct) as Table 2. We report pass@50 concept-guided differences across CG/AG size pairs.

	AG				
	1.5B	3B	7B	14B	32B
Baseline pass@50	18.4	16.6	11.0	12.6	11.8
CG / AG	**Difference (Concept - Baseline) pass@50**				
1.5B	+0.8	+1.2	+4.1	+0.5	+1.9
3B	+0.7	+0.7	+5.3	+1.0	+3.1
7B	-0.1	+2.5	+4.2	-1.0	+4.3
14B	+1.1	-0.3	+5.4	+2.3	+2.0
32B	+0.2	-1.7	+6.8	+1.0	+4.7

D.6 Why Naive Resampling Is Nonzero on the "0%" Held-Out Set

We build the held-out set by drawing $B = 128$ answers per problem with the answer generator and keeping the problems with zero correct samples. This is a finite-sample estimate of the AG success probability p, not a guarantee that $p = 0$. A problem with small but nonzero p passes the filter with probability $(1-p)^{128}$, yet an independent fresh draw of 128 answers solves it at least once, with probability $1-(1-p)^{128}$.

Empirically, the naive baseline reaches pass@128 $\approx$ 19% on this filtered set (Table 3). Because this estimate is based on only (128) fresh evaluation rollouts per problem, it may be sensitive to the particular sampling draw. We therefore re-estimate it below using a much larger rollout allocation.

Re-estimating the naive pass@128. The main-table pass@128 is computed with the unbiased estimator (Chen et al., 2021) on per-problem success counts s_i out of $B = 128$ rollouts. At $k = B = 128$ this estimator degenerates to the plain solve rate $\mathbb{1}[s_i \geq 1]$, so it carries the full Bernoulli seed noise and could in principle over- or under-shoot the true value. To estimate this variance and make sure our results are well beyond the seed variance of the naive sampling baseline, we draw a fresh batch of 1280 answers per problem, generated as 10 independent seeds of 128 answers each (and independent of the filtering seed), and re-estimate the naive pass@128 with three estimators (Table 7): (i) the unbiased estimator with $k = 128 < B = 1280$, applied to all 1280 pooled rollouts, which analytically averages over rollout noise (equivalent to averaging over all $\binom{1280}{128}$ subsets); (ii) a Monte Carlo estimate that draws 10,000 random 128-rollout subsets and reports the mean and standard deviation of the plain solve rate; and (iii) a direct seed-level estimate that reports the mean and standard deviation across the 10 independent seeds.

The unbiased estimator gives 19.46%, and the Monte Carlo estimate agrees closely at 19.47%, with a standard deviation of 0.94 points across random 128-rollout subsets. The mean across the 10 independent seeds is likewise 19.44%, with a standard deviation of 1.43 points across seeds. Naive repeated sampling therefore recovers $\approx$ 19.5% of the "0%" held-out problems on a fresh seed.

Table 7: Re-estimated naive pass@128 on the "0%" held-out set (fresh batch of $B = 1280$ rollouts per problem).

Estimator	pass@128 (%)	std (%)
Unbiased (all 1280 rollouts, $k = 128$)	19.46	—
Monte Carlo (10,000 random 128-subsets)	19.47	0.94
Independent seeds (10×128 rollouts)	19.44	1.43

D.7 Extended Pass@k Curves and Confidence Intervals

The main table (Table 3) reports point estimates at $k \in \{1, 64, 128\}$. Here we report the full dependence on the rollout allocation together with estimation uncertainty, in two complementary forms: full pass@k curves with confidence bands (Figure 5), and per-method gains over the naive baseline with paired-bootstrap confidence intervals (Table 8).

Full curves with confidence bands. Figure 5 plots pass@k against the rollout allocation k for every method, with one panel per dataset (DeepMath held-out hard, Omni-MATH 2 OOD) and a shaded 95% bootstrap confidence band around each curve. This shows the entire dependence on k—superseding the three snapshot columns of the main table—together with estimation uncertainty, and makes the method ordering, and where bands overlap, visible at a glance.

Gains over the naive baseline. Because all methods are scored on the same held-out problems, the quantity of interest is the per-problem gain over naive repeated sampling rather than the absolute pass@k. Table 8 reports, for each method, the difference Δpass@k relative to naive at $k \in \{1, 64, 128\}$ with a 95% paired-bootstrap interval; an interval that excludes 0 indicates a statistically significant improvement. On DeepMath, every method improves significantly over naive at all allocations, and the gains grow sharply with k—from under two points at pass@1 to double digits at pass@128—the signature of better *exploration* rather than single-shot accuracy. The ordering is consistent across allocations, with the trained max-of-mean policy strongest ($+20.2$ points at pass@128), followed by max-of-max ($+16.3$), the untuned 32B CG ($+14.8$), the untuned 7B CG ($+9.9$), and the generic prompt-modification baseline ($+3.6$). On the Omni-MATH 2 OOD set the gains are smaller but the concept-guided methods remain significant at every allocation (max-of-mean $+7.3$ points at pass@128, intervals excluding 0), whereas the generic prompt-modification baseline is statistically indistinguishable from naive throughout (e.g. -0.07 $[-1.88, +1.74]$ at pass@128). Problem-specific concepts, and especially the trained max-of-mean policy, thus yield significant gains that generic prompt perturbation does not, and these gains persist out of distribution.

Estimation procedure. For each problem i and each method, we generate $B = 128$ answer rollouts. For concept-guided methods, we distribute these rollouts as evenly as possible across the M concepts and pool the resulting candidates. Let s_i denote the total number of correct answers in this pooled set. We then compute the standard per-problem estimate $\widehat{\text{pass@}k}_i = 1 - \binom{B-s_i}{k} / \binom{B}{k}$ and report pass@k as its mean over the corresponding held-out set.

Confidence intervals come from a bootstrap over problems: we resample the $\sim$1000 problem indices with replacement, recompute the mean, repeat 10,000 times, and take the 2.5/97.5 percentiles. For the gain of one method over another we use a *paired* bootstrap—resampling problems once and taking the difference of the two means on that same resample—since both methods are evaluated on identical problems. (At $k = B = 128$ the estimator degenerates to the plain solve rate; see Appendix D.6 for when extra rollouts, rather than seeds, are needed to stabilize it.)

Table 8: Gain over naive repeated sampling, Δpass@k (%), with 95% paired-bootstrap confidence intervals (over held-out problems). Cells are formatted as Δ [lo, hi]; an interval excluding 0 indicates a significant improvement.

Method (vs. naive)	Δpass@1	Δpass@64	Δpass@128
DeepMath (held-out hard)			
Generic prompt modification	$+0.09$ $[+0.05, +0.13]$	$+3.18$ $[+1.45, +4.87]$	$+3.60$ $[+0.80, +6.30]$
Untuned CG (7B)	$+0.54$ $[+0.40, +0.68]$	$+9.27$ $[+7.15, +11.32]$	$+9.90$ $[+6.90, +12.80]$
Untuned CG (32B)	$+0.59$ $[+0.47, +0.71]$	$+12.42$ $[+10.29, +14.54]$	$+14.80$ $[+11.70, +17.90]$
max-of-max	$+1.25$ $[+1.02, +1.50]$	$+15.31$ $[+12.93, +17.69]$	$+16.30$ $[+13.00, +19.60]$
max-of-mean	$+1.95$ $[+1.59, +2.33]$	$+18.29$ $[+15.91, +20.73]$	$+20.20$ $[+17.00, +23.40]$
Omni-MATH 2 (OOD)			
Generic prompt modification	$+0.01$ $[-0.02, +0.04]$	$+0.12$ $[-0.99, +1.21]$	-0.07 $[-1.88, +1.74]$
Untuned CG (7B)	$+0.14$ $[+0.09, +0.19]$	$+2.83$ $[+1.63, +4.03]$	$+3.69$ $[+1.74, +5.57]$
Untuned CG (32B)	$+0.13$ $[+0.09, +0.18]$	$+3.37$ $[+2.14, +4.62]$	$+3.98$ $[+2.03, +5.93]$
max-of-max	$+0.13$ $[+0.09, +0.18]$	$+3.23$ $[+1.98, +4.45]$	$+3.98$ $[+2.03, +5.93]$
max-of-mean	$+0.30$ $[+0.23, +0.39]$	$+5.94$ $[+4.60, +7.35]$	$+7.31$ $[+5.28, +9.41]$

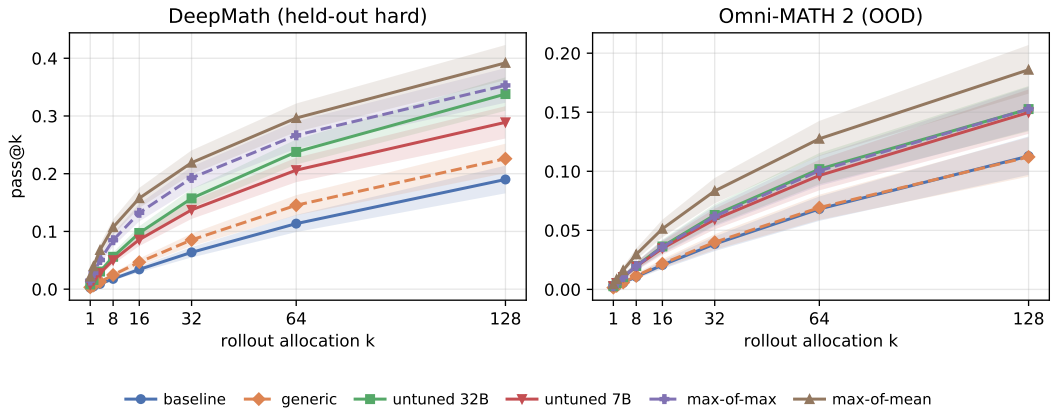


Figure 5: Pass@k vs. rollout allocation k for all methods, one panel per dataset (DeepMath held-out hard, Omni-MATH 2), with shaded 95% bootstrap confidence bands (bootstrap over held-out problems).

Trained 7B CG vs. untuned 32B CG. A central claim of Section 6 is that the *trained* 7B concept generator beats the *untuned* 32B one, despite steering the same frozen answer generator with far fewer parameters. We test this gap directly with the same paired bootstrap over held-out problems, taking the per-problem difference of the two unbiased pass@k estimates on identical problems (Table 9). The trained max-of-mean CG significantly outperforms the untuned 32B CG at every allocation on both datasets (all 95% intervals exclude 0) by $+5.4$ $[+2.4, +8.6]$ points at pass@128 on DeepMath and $+3.3$ $[+1.2, +5.4]$ on Omni-MATH 2.

Table 9: Direct gain of the RL trained 7B CG over the untuned 32B CG, Δpass@k (%), with 95% paired-bootstrap confidence intervals (over held-out problems). Cells are formatted as Δ [lo, hi]; an interval excluding 0 indicates a significant improvement.

Method (vs. untuned 32B CG)	Δpass@1	Δpass@64	Δpass@128
DeepMath (held-out hard)			
max-of-max	$+0.67$ $[+0.46, +0.88]$	$+2.88$ $[+0.67, +5.11]$	$+1.50$ $[-1.60, +4.60]$
max-of-mean	$+1.37$ $[+1.05, +1.70]$	$+5.87$ $[+3.70, +8.11]$	$+5.40$ $[+2.40, +8.60]$
Omni-MATH 2 (OOD)			
max-of-max	$+0.00$ $[-0.04, +0.05]$	-0.14 $[-1.46, +1.15]$	$+0.00$ $[-2.03, +2.03]$
max-of-mean	$+0.17$ $[+0.10, +0.26]$	$+2.57$ $[+1.16, +4.00]$	$+3.33$ $[+1.16, +5.43]$

Effect of RL training (trained vs. untuned 7B CG). Since the untuned and trained 7B CG are the *same* model before and after RL, their difference isolates the effect of training itself. Under the same paired bootstrap (Table 10), training yields significant gains at every allocation on DeepMath (max-of-mean $+10.3$ $[+7.2, +13.4]$ points at pass@128; max-of-max $+6.4$ $[+3.4, +9.4]$). Out of distribution on Omni-MATH 2, max-of-mean stays significant at all allocations ($+3.6$ $[+1.5, +5.6]$ at pass@128), whereas max-of-max is indistinguishable from the untuned 7B CG (intervals include 0). Training therefore improves concept quality beyond what better prompting and filtering achieve, and only the max-of-mean objective carries this improvement out of distribution.

D.8 Diversity of the Generated Reasoning

A central claim of our approach is that concept guidance improves *exploration*: it should yield genuinely different solution attempts rather than the near-duplicates produced by decoding noise alone. We test this directly by measuring the semantic diversity of the generated reasoning on the DeepMath val-hard held-out set.

Setup. For each problem we collect the model's rollouts and embed only the chain-of-thought excluding the problem statement and the final answer, so that the score reflects the diversity of the *reasoning* itself rather than shared prompt text or a common answer. We use Qwen3-Embedding-8B

Table 10: Gain of the RL-trained 7B CG over the untuned 7B CG (the same model before training), Δpass@k (%), with 95% paired-bootstrap confidence intervals (over held-out problems). Cells are formatted as Δ [lo, hi]; an interval excluding 0 indicates a significant improvement.

Method (vs. untuned 7B CG)	Δpass@1	Δpass@64	Δpass@128
DeepMath (held-out hard)			
max-of-max	$+0.72$ $[+0.48, +0.96]$	$+6.04$ $[+3.85, +8.27]$	$+6.40$ $[+3.40, +9.40]$
max-of-mean	$+1.42$ $[+1.08, +1.78]$	$+9.03$ $[+6.77, +11.39]$	$+10.30$ $[+7.20, +13.40]$
Omni-MATH 2 (OOD)			
max-of-max	0.00 $[-0.06, +0.06]$	$+0.40$ $[-0.90, +1.68]$	$+0.29$ $[-1.74, +2.32]$
max-of-mean	$+0.17$ $[+0.09, +0.25]$	$+3.11$ $[+1.74, +4.49]$	$+3.62$ $[+1.52, +5.64]$

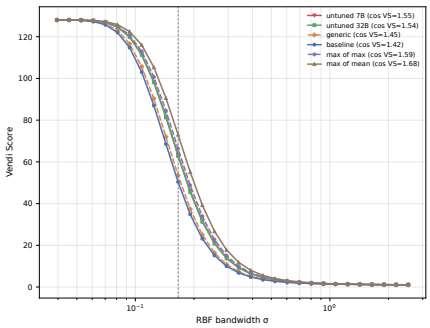


Figure 6: Average Vendi Score (effective number of distinct rollouts) of the generated chains of thought vs. the RBF kernel bandwidth σ, per method, averaged over the DeepMath val-hard held-out set. The methods are maximally separated at $\sigma = 0.166$ (spread $= 22.7$), the bandwidth used in Table 11.

(Zhang et al., 2025) as the sentence embedder. On the resulting set of embeddings we compute the Vendi Score (VS) (Friedman & Dieng, 2023) with an RBF kernel $k(x, y) = \exp\left(-\|x-y\|^2/(2\sigma^2)\right)$. The VS is the exponential of the Shannon entropy of the eigenvalues of the normalized kernel matrix and can be read as the *effective number of distinct rollouts*, so a higher VS means more diverse reasoning. We compute VS per problem and average over the held-out set.

Kernel bandwidth. The bandwidth σ sets the scale at which two chains of thought count as similar, so we sweep it and plot the average VS of each method against σ (Figure 6). We summarize each method at the bandwidth that maximizes the spread between methods (the difference between the largest and smallest average VS), $\sigma = 0.166$, where the spread reaches 22.7 points and the methods are best separated.

Results. Table 11 reports the average VS at this bandwidth, and diversity increases exactly in the order our exploration story predicts. Naive repeated sampling is the least diverse (50.4), and perturbing the prompt with generic mathematical text barely helps (53.5). Conditioning on sampled concepts raises diversity substantially, whether the concepts come from the untuned 7B or 32B generator (63.3 and 62.9, essentially tied). Training the concept generator increases it further: max-of-max reaches 66.3, and our best method, max-of-mean, is the most diverse at 73.0—a 45% increase in effective distinct rollouts over the naive baseline. This ordering mirrors the pass@k gains in Table 8: methods that explore more diversely also solve more problems, supporting our claim that the gains come from broader exploration rather than prompt perturbation alone.

D.9 Effect of Longer Training

Our main results (Table 3) train the concept generator for 200 steps. Because each step is dominated by the $G \times B$ answer-generator rollouts needed to score the sampled concept trajectories (Section 7), we check whether this training length leaves performance on the table by continuing training well past it: max-of-max up to 750 steps and max-of-mean up to 900 (Figure 7).

Table 11: Average Vendi Score of the generated chains of thought (embedded with Qwen3-Embedding-8B, RBF kernel at bandwidth $\sigma = 0.166$) on the DeepMath val-hard held-out set. Higher indicates more diverse reasoning. Best in **bold**, second best in *italic*.

Method	Vendi Score
Naive baseline	50.38
Generic prompt modification	53.54
Untuned CG (7B)	63.33
Untuned CG (32B)	62.85
max-of-max	*66.32*
max-of-mean	**73.04**

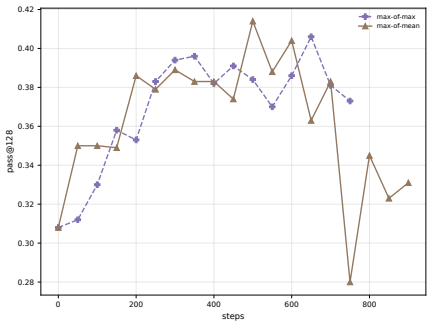


Figure 7: Held-out pass@128 on the DeepMath val-hard set over the course of training, for the two reward-aggregation objectives (max-of-max up to 750 steps, max-of-mean up to 900).

Performance largely saturates after the first few hundred steps. Both objectives climb steeply over the first $\sim$200 steps and then plateau in a noisy band around 0.38–0.40 pass@128, with no sustained improvement thereafter. They differ mainly in how they reach this plateau: max-of-mean rises faster and is ahead at 200 steps (0.386 vs. 0.353 pass@128), whereas max-of-max catches up by $\sim$ 300 steps, after which the two are indistinguishable within the run-to-run noise. The slower early progress of max-of-max is expected: its trajectory reward is binary (1 if any concept in the trajectory yields a correct answer and 0 otherwise) which, compounded by sampling noise, makes credit assignment harder and the learning signal less informative than the graded per trajectory success rate used by max-of-mean. Neither ever pulls clearly above the level already reached around 200 steps.

The two objectives do differ in late-training stability. max-of-max stays within the plateau band through 750 steps, whereas max-of-mean grows increasingly volatile and collapses sharply around step 750 (pass@128 dropping to $\approx$0.28), recovering only partially afterwards. This collapse reflects a training instability rather than genuine overfitting: around the same point the policy entropy rises sharply and the concept generator increasingly emits over-long trajectories that hit the generation-length limit and are truncated before producing a stop token (the truncation rate climbs to $\approx 60\%$ near step 720), alongside a growing KL divergence from the reference policy. Such truncated concept trajectories are malformed and score poorly downstream, dragging pass@128 down. Together, these observations justify our 200-step schedule: it captures essentially all of the attainable gain at a fraction of the compute, and stopping early avoids the instability that eventually affects max-of-mean.

D.10 Qualitative Examples: Concepts Before and After Training

To illustrate qualitatively how training reshapes the generated concepts, we show three held-out problems for which concepts from the untuned 7B generator fail to elicit a correct AG answer, whereas concepts from the trained max-of-mean generator succeed (Examples 2 to 4). Ground-truth answers are shown only for reference and are not included in either the concept generator or answer generator prompts.

For each problem, we show three concepts from the untuned generator followed by three from the trained generator. In these examples, the trained concepts are more concrete and problem specific, providing a particular substitution, construction, or explicit line of attack, whereas the untuned concepts more often state generic strategies or name a relevant theorem without a usable plan.

Example 2: Distributing balls with capacity constraints

Problem. Find the generating function for the number of ways to distribute n identical juggling balls to five different jugglers, where each juggler receives at most seven balls. Using this generating function, determine the number of ways to distribute 25 balls.
Ground truth: 926.

Untuned 7B CG

- Use the multinomial theorem to expand the generating function.
- Consider the stars and bars method with constraints to derive the generating function.
- Use the principle of inclusion-exclusion to count the valid distributions.

Trained max-of-mean

- Use a generating function approach. We can construct a generating function for the number of ways to distribute the balls under the given constraint. Each juggler can have 0, 1, 2, 3, 4, 5, 6, or 7 balls, so the generating function for one juggler is $1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$. Since there are five jugglers, the generating function for the distribution of all balls is the fifth power of this function.
- Explore a combinatorial approach. Instead of using generating functions, consider a combinatorial argument. The problem can be translated into finding the number of non-negative integer solutions to the equation $x_1 + x_2 + x_3 + x_4 + x_5 = 25$ with $0 \leq x_i \leq 7$. This can be approached using the principle of inclusion-exclusion.
- Consider a recursive approach. Define a recursive function $f(n, k)$ that represents the number of ways to distribute n balls among k jugglers such that each juggler gets at most 7 balls. Use the recursive relation $f(n, k) = \sum_{i=0}^{7} f(n - i, k - 1)$ with base cases $f(0, k) = 1$ and $f(n, 0) = 0$ for $n > 0$.

Example 3: Summing repeated integrals

Problem. For non-negative integers n, define a function $h_n(x)$ as follows:

$$h_n \colon [0, 1] \to \mathbb{R}, \quad h_n(x) = \int_0^x h_{n-1}(t)dt \quad (n > 0), \quad h_0(x) = x^2$$

Compute the sum $\sum_{n=0}^{\infty} h_n(x)$. Express your answer in terms of x.
Ground truth: $2(e^x - x - 1)$.

Untuned 7B CG

- Recognize the pattern in integrals and derivatives.
- Use the exponential function series expansion.
- Apply induction to confirm the pattern of $h_n(x)$.

Trained max-of-mean

- Analyze the recursive relationship. We can start by computing the first few terms of the sequence explicitly to identify a pattern. For $n = 1$, we have:

$$h_1(x) = \int_0^x h_0(t)\,dt = \int_0^x t^2\,dt = \frac{x^3}{3}.$$

  For $n = 2$:

$$h_2(x) = \int_0^x h_1(t)\,dt = \int_0^x \frac{t^3}{3}\,dt = \frac{x^4}{12}.$$

  This suggests a general form for $h_n(x)$ might be of the form $\frac{x^{n+2}}{n!\cdot k_n}$ for some sequence k_n.

- Use generating functions. Define a generating function $H(x, y) = \sum_{n=0}^{\infty} h_n(x)y^n$. Then, the recursive relation can be translated into a functional equation for $H(x, y)$. For example:

$$H(x, y) = h_0(x) + y \int_0^x H(t, y)\,dt.$$

- Check for a closed-form solution using differentiation. Differentiate the recursive relation with respect to x and see if a pattern emerges. For example, if we differentiate $h_n(x) = \int_0^x h_{n-1}(t)\,dt$ with respect to x, we get:

$$h_n'(x) = h_{n-1}(x).$$

  This might help in finding a relation between $h_n(x)$ and $h_{n-1}(x)$.

Example 4: Constant-speed elevators (relative motion)

Problem. Four elevators in a skyscraper, each a different color (red, blue, green, and yellow), move at constant speeds in various directions. The following events were observed: at 36 seconds the red elevator caught up with the blue one; at 42 seconds the red elevator passed the green one; at 48 seconds the red elevator passed the yellow one; at 51 seconds the yellow elevator passed the blue one; at 54 seconds the yellow elevator caught up with the green one. Determine the time, in seconds from the start, when the green elevator will pass the blue one.
Ground truth: 46.

Untuned 7B CG

- Determine the relative speeds between pairs of elevators.
- Construct a system of equations based on the observed events.
- Plot the positions of the elevators as functions of time.

Trained max-of-mean

- Introduce a coordinate system where one elevator is at rest to simplify the problem. Choose the blue elevator as the reference frame. Rewrite the positions and speeds of the other elevators relative to the blue elevator. This can simplify the equations since the blue elevator's position is constant in this frame.
- Solve for the initial positions and velocities of the elevators. Given that the red elevator catches up with the blue at 36 seconds, we can use this event to write an equation for the initial positions and velocities. Similarly, use the other events to form a system of linear equations to solve for the unknowns.
- Consider the scenario where the green elevator passes the blue elevator. Set up an equation where the position of the green elevator equals the position of the blue elevator and solve for the time t.