MEASURING AND MITIGATING SOLUTION MODE COLLAPSE IN RLVR

Anonymous authors
Paper under double-blind review

ABSTRACT

A language model (LM) can usually answer the same question in more than one way, but reinforcement learning with verifiable rewards (RLVR) is indifferent to which correct answer a model produces. A solution will earn the same reward whether it is the thousandth copy of a familiar answer or one the model has never produced before. Yet, there is potential value in having the model retain multiple correct solutions as it is trained. For instance, multiple modes may give users a choice and provide problem-solving strategies that improve overall model performance. Here, we introduce MODEBENCH, a benchmark of multi-solution tasks in which the verifier returns both correctness and mode discovered. We then use MODEBENCH to measure how solution diversity changes under RLVR post-training. We find that RLVR post-training concentrates probability onto fewer correct modes even as accuracy holds or improves, and moreover, that frontier models are already highly concentrated. We then introduce our solution, Re:Max, which stores one verified example per discovered mode in a replay buffer and trains on those stored modes uniformly. A solution found once is, therefore, practiced as often as one found repeatedly. Across three model scales, two RL objectives, and harder task constructions, replay improves both how often a policy succeeds and how many different ways it can succeed.

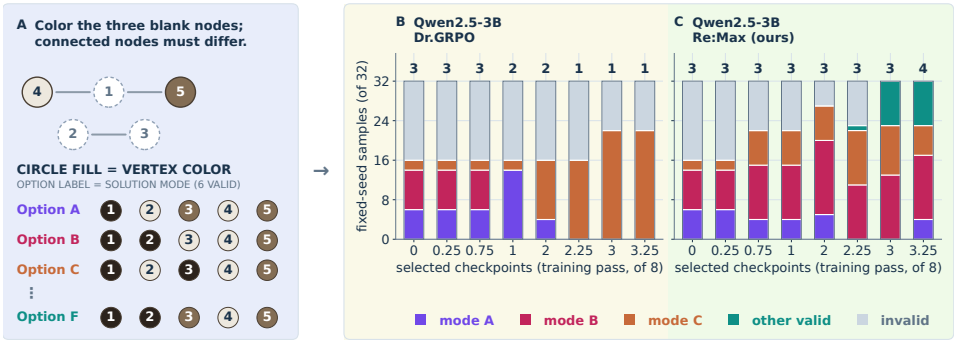

Figure 1: **Motivation.** Consider the `Graph` coloring problem shown in **A** (left), with six valid solutions. We evaluate Qwen2.5-3B across training, tracking the number and diversity of correct solutions. At each training pass, we show 32 sampled questions, eight from each of four matched runs. Training with Dr. GRPO (**B**) improves accuracy and leads to a single solution dominating the output. In contrast, Re:Max (**C**) increases both accuracy and diversity of correct solutions.

1 INTRODUCTION

Training and evaluation of reinforcement learning with verifiable rewards (RLVR) algorithms are driven entirely by model accuracy. However, what makes models useful at deployment depends on having different candidates to choose from, i.e. test-time search and reranking (Brown et al., 2024; Snell et al., 2024; Cobbe et al., 2021; Wang et al., 2023). As RLVR improves accuracy, what happens to the diversity of correct solutions?

🤗 **ModeBench dataset and models** **Code repository** *Links anonymized for review.*

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.

RLVR commonly uses automatically verifiable outcome rewards (Guo et al., 2025). These rewards do not distinguish among different correct solutions. In binary-verifier formulations, all verified-correct solutions receive the same task reward, regardless of which solution mode they belong to. Consequently, improving correctness does not determine how the model should weight alternative correct solutions. Fig. 1 shows what this can look like on a graph-coloring problem with six valid answers: under Dr.GRPO (Liu et al., 2025), the policy becomes more accurate while its correct outputs concentrate onto a single mode.

Mode collapse, or narrowing of the output space, has been observed in alignment (Kirk et al., 2024; Cui et al., 2025), creative generation (Doshi & Hauser, 2024), brainstorming and ideation (Anderson et al., 2024), public argument (Kim et al., 2026), and LM-assisted writing (Abdulhai et al., 2026). Studying mode collapse, however, requires deciding when two correct outputs constitute genuinely different solutions. Approaches based on token-level diversity (Li et al., 2016; Zhu et al., 2018) are computationally inexpensive, but can mistake superficial reformulations for distinct solutions. Semantic grouping methods (Farquhar et al., 2024; Zheng et al., 2023) instead use a second model to judge whether outputs are meaningfully different. As-judge approaches have both (1) an additional inference cost per generation and (2) rely upon the judge to determine diversity. To avoid this tradeoff, we study settings where a solution identity can be determined directly by an executable verifier. This gives us an inexpensive, deterministic, verifiable measure of diversity that is insensitive to superficial wording.

Here, we investigate the prevalence and impact of mode collapse in problem-solving tasks under RLVR post-training. Our contributions are threefold:

1. MODEBENCH: We contribute a benchmark for empirically studying mode collapse in tasks allowing for multiple correct solutions, ranging from code evaluation to constrained planning. MODEBENCH provides both output correctness and a unique identity for each solution mode. The benchmark spans five executable domains, each with five difficulty levels (Sec. 2).
2. We conduct comprehensive experiments on MODEBENCH across 4 open-weight model families and 7 frontier models. Across open-weight models and at 3 model scales, RLVR substantially reduces solution-mode diversity across domains under both Dr.GRPO and GRPO. Further, in inference, all frontier models we measured reached fewer than 1.5 effective modes out of up to 355 unique solutions (Sec. 5.1 and App. N.1).
3. We propose *mode replay*, which stores one verified exemplar per mode and rehearses all modes equally. Added to Dr.GRPO (Liu et al., 2025), it raises accuracy and pairwise correct mode diversity across 3 model scales and all 5 MODEBENCH domains. Across the five-domain aggregate, our replay methods outperform 8 RLVR comparators on both correctness and conditional solution diversity. Added to MaxRL (Tajwar et al., 2026) (which we dub as Re:Max), it adds diversity and accuracy across MODEBENCH difficulty levels (Sections 3 and 5.2–5.3).

Overall, our results quantify the severity of mode collapse and suggest that incorporating mode replay as part of RLVR post-training can lead to both increased diversity of solutions and improved model performance.

2 MODEBENCH: MEASURING SUCCESSFUL ALTERNATIVES

Building a benchmark to study solution mode collapse is challenging for two reasons. First, each problem must allow for several valid discrete solutions. Second, given a model's re-

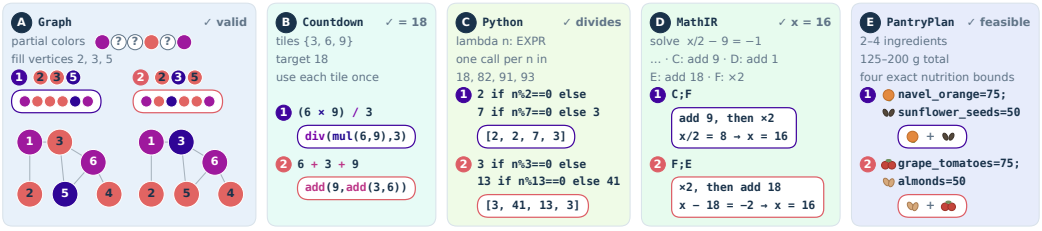

Figure 2: **MODEBENCH comprises questions with many acceptable solutions across domains.** (1) and (2) identify different solution modes across the five domains, which include Graph coloring, a normalized computation tree, executable Python code, algebraic problems and constraint optimization.

sponse, we must be able to determine which solution mode was reached. For example, in Fig. 2D, we show algebra problems in our benchmark. In this example, the same algebraic solution can be reached through two different series of steps.

Ensuring these two properties hold is domain-specific. Fig. 2 illustrates a subset of solution modes across the considered domains; e.g., in `Countdown` and `MathIR`, modes mean the model reached the same answer by a different route, whereas in `Graph` and `PantryPlan`, modes mean a different valid answer altogether. We provide the certified or enumerated solution support used for each domain in Table 2. Refer to App. B and App. C for per-domain rules and prompts for dataset generation.

For each domain, we provide a verifier that determines both correctness and solution-mode identity. Precisely, for a prompt x with specification s_x, the verifier maps an output y either to failure $\perp$ or to a solution mode $V(y, s_x) \in \mathcal{C}_x$. Two accepted outputs are the same mode if and only if their verifier outputs agree. The learner observes only modes for responses it generates and is never given the complete set $\mathcal{C}_x$ or its size. Some domains additionally provide certified or enumerated support used for validation and analysis; these catalogues need not exhaust every solution accepted by the verifier.

2.1 Success and Verified-Mode Diversity

To measure performance on ModeBench, we must consider two distinct properties: how often the output is accurate, and, given that it is right, which of the available right answers has been given. A benchmark reporting only accuracy cannot capture the failure we study.

The obvious way to measure modal diversity is to count distinct correct answers in K samples. We define P as the chance a sample is correct and q_c as how that correct probability is divided among the valid answers:

$$\underbrace{\Pr[D_K > 0]}_{\texttt{pass@}K} = 1 - (1-P)^K, \qquad \underbrace{\mathbb{E}[D_K]}_{\texttt{distinct@}K} = \sum_c [1 - (1 - Pq_c)^K], \tag{1}$$

where D_K counts the distinct correct answers. However, both metrics increase with P, and thus a policy that is right more often may have higher `pass@K` and `distinct@`K while producing less diverse solutions.

We therefore propose an alternative metric, denoted as pairwise correct-mode diversity:

$$\texttt{PCMD} = 1 - \sum_c q_c^2 = \Pr[Z_1 \neq Z_2 \mid \text{both responses verified}], \tag{2}$$

measuring the joint probability of different solution modes given two correct responses. A `PCMD` of zero thus implies every correct answer belongs to the same solution mode, and an even spread over m yields a `PCMD` of $1 - 1/m$, e.g., if answers spread evenly across two possible solution modes, then the `PCMD` is $1/2$. We note that when infinite solution modes are possible, the maximum `PCMD` converges to 1, and that when the model outputs no correct solutions, we cannot calculate the metric. We only calculate `PCMD` when a model produces a minimum number of correct solutions; App. A gives that threshold.

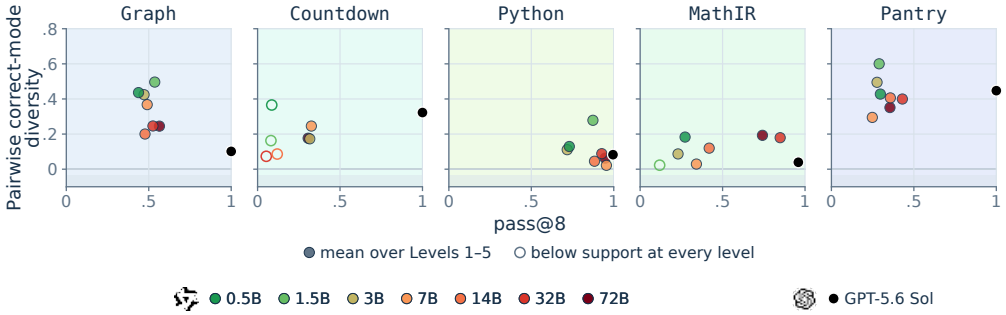

Figure 3: **Higher performance is not associated with increased solution mode diversity.** We show the relationship between a performance-related metric (`pass@8`; x-axis) and our proposed solution mode diversity metric (`PCMD`; y-axis) for GPT-5.6 Sol (in black), and Qwen2.5-Instruct models of various sizes (where parameter count is indicated by color). Note that accuracy and model size are not correlated with solution mode diversity.

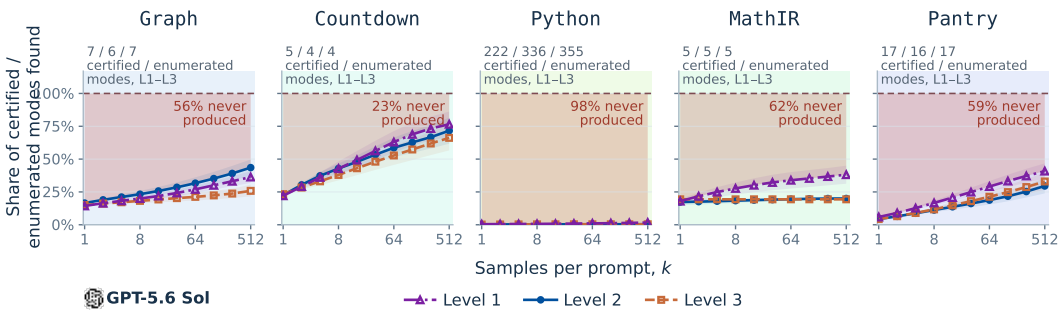

Figure 5: **Asking the model more times may not recover all certified or enumerated modes.** We test directly whether a larger sampling budget recovers the known modes. GPT-5.6 Sol, with medium reasoning on, answers 32 problems per domain and level with 512 responses each. Curves are distinct verified modes in random k-subsets as a share of the enumerated or certified modes available for those problems, and the shaded band is what no level ever produced. Shading around lines is the pointwise 95% CIs using a bootstrap over problems.

Fig. 3 shows how `PCMD` captures diversity of responses independently of accuracy. GPT-5.6 Sol has near-perfect `pass@8` for all five tasks considered, yet much weaker models in the Qwen2.5-Instruct family have higher `PCMD`. For the context of this work, we define a *solution mode collapse* as a decline in `PCMD` during RLVR.

2.2 Impact of task difficulty on mode collapse

We construct ModeBench with multiple difficulty levels so that we can study how solution-mode diversity changes as problems become harder, while keeping the task-defined solution mode fixed. For each domain, we include five difficulty levels, e.g. for our `Graph` domain we allow more hidden vertices, and for `Countdown` we give the model more operands. Each difficulty level holds constant (1) our definition of mode, (2) our approximate distribution of modes, (3) our verifier, and (4) the sample size of 384 train and 128 test problems. To ensure (2) holds, and moreover, that the problems we sample are in fact more challenging, we fit each level with consideration to `pass@1` and `pass@8` of a reference model, e.g. Level 4 was tuned such that a Qwen2.5-Instruct-7B model has the same `pass@1` and `pass@8` scores as a Qwen2.5-Instruct-0.5B on Level 1.

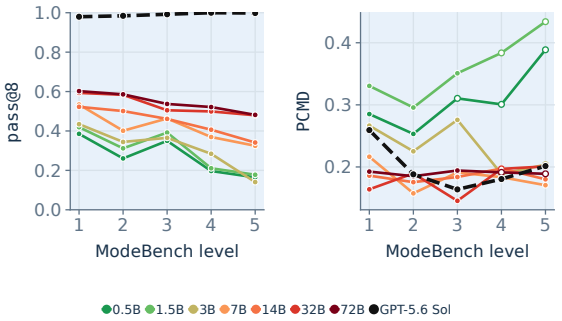

Figure 4: **Harder problems of the same domain exhibit fewer solution modes.** When we ask the Qwen2.5-Instruct family and GPT-5.6 Sol increasingly difficult questions from harder levels of ModeBench, `pass@8` and `PCMD` both decline. For our benchmark, as problems get harder, solution mode collapse worsens.

As task difficulty increases, solution-mode diversity generally declines, including in settings where correctness remains near saturation (Fig. 4). This is most noticeable when frontier models are evaluated. Even though correctness is saturated, we are able to use `PCMD` to isolate solution mode collapse (App. N.1), and this pattern appears across ModeBench. For example, frontier models answer .991 of questions from Level 1–3 correctly under normalized grading, but their mean `PCMD` is only .232, below the .570 of the most diverse sub-2B model, which answers .273 of its prompts.

2.3 Concentration in frontier models

A common fix for solution mode collapse is to increase the sampling budget, as repeated sampling can substantially increase solution coverage at test time (Brown et al., 2024; Yue et al., 2025). But, across ModeBench, larger sampling budgets remain far below the enumerated or certified solution support in several domains. For example, all 1,024 GPT-5.6 Sol draws on the

128 Level-3 `Graph` problems verify, yet eight draws yield just 1.16 distinct colorings, and 95.1% of correct pairs repeat one (App. N.1). Fig. 5 follows MODEBENCH at three levels out to 512 draws per problem, and the curves flatten far below the modal set: an LM that has settled on two colorings does not find the rest even after being asked five hundred times.

`Python` is the extreme case. Its problems admit several hundred distinct return vectors on average, and five hundred draws surface between one and four solutions, under two percent of all possible solutions, and under one percent at Levels 2 and 3. Each doubling of the sampling budget buys a fraction of one mode, against gaps of several modes everywhere and several hundred in `Python`. This problem becomes worse as we increase problem difficulty. All 7 frontier models tested narrow in `PCMD` from Level 1 to Level 3 while staying near-perfect on correctness (Table 13, App. N.1).

We similarly check our local models over larger samplings, and we find quadrupling to $K = 32$ draws per problem moves control `PCMD` by at most .010, and `MathIR` at $K = 64$ draws is as concentrated as at $K = 8$ (Apps. I and J), while disabling reasoning lowers correctness and verified-mode counts in every deployment tested, though its `PCMD` effect differs by deployment (Table 16, App. N.3).

We do, however, note that cheap inference interventions do exist. For example, we find that because different models collapse onto different answers, mixing responses from two independent models covers more verified solution modes than either alone (App. K.1). Additionally, raising inference temperature can expose more sampled solution modes, sometimes at a cost in accuracy (Table 17, App. N.4).

3 RE:MAX: LEARN FROM NEW SAMPLES AND REMEMBER SUCCESSES

3.1 THEORETICAL INTUITION FOR MODE COLLAPSE

Solution mode collapse follows from two properties of the objective: 1) the binary verifier assigns the same reward to every correct answer, so the objective provides no incentive to preserve one correct mode over another; and 2) an on-policy update can reinforce only the modes that appear in its fresh rollout samples, which are drawn according to the policy's current preferences. Together, these properties create a positive feedback loop, where a mode with even a small early advantage is sampled more often, receives more reinforcement, and therefore becomes even more likely to be sampled in subsequent updates. Figure 6 illustrates this process.

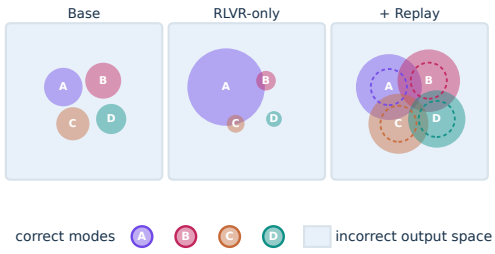

Figure 6: **RLVR early "winner" (A) becomes the predominant way to solve.** The base model supports several modes. RLVR-only training concentrates on one mode. Our replay approach keeps every buffered mode & leaves room for more.

Recent work formalizes this positive-feedback effect, showing that on-policy RL can amplify small probability differences among equally rewarded outcomes into "winner-takes-all" dynamics (Sinha et al., 2026; Lochab et al., 2026). We extend existing analysis to the verified solution-mode setting; App. O gives the formal results and assumptions. First, under an idealized solution-mode model over verifier-defined modes, the same winner-takes-all dynamic applies. Second, we show that binary rewards alone do not force mode concentration; the effect depends on their interaction with the optimization geometry. Third, we characterize why collapse becomes difficult to reverse under on-policy training. We show that as a solution mode becomes rare, both its probability of appearing in a rollout and its expected gradient contribution approach zero.

3.2 REPLAYING DISTINCT VERIFIED SUCCESSES

These results motivate a simple intervention. Once a correct solution mode has been discovered, continued training should not depend on the current policy re-discovering it. We therefore use experience replay to retain verified successes and continue training on them (Lin, 1992; Mnih et al., 2013; Rolnick et al., 2019). In our idealized fixed-buffer analysis, replay prevents a stored mode's

probability from vanishing: once a mode is stored in the buffer, its probability remains bounded away from zero throughout subsequent training under the assumptions of Theorem O.4.

For each training prompt x, we maintain an initially empty, bounded buffer $\mathcal{B}_x$. The first time a verified mode appears, we store one exemplar e_c (subsequent rediscoveries do not modify the buffer). A deterministic schedule revisits nonempty buffers throughout training. For an exemplar of token length $|e_c|$, the replay loss is

$$L_{\text{replay}}(x) = -\frac{1}{|\mathcal{B}_x|} \sum_{c \in \mathcal{B}_x} \frac{\log \pi_\theta(e_c \mid x)}{|e_c|}. \tag{3}$$

Uniform weighting breaks the link between discovery frequency, how often a mode appears in fresh rollouts, and rehearsal frequency, how often a stored mode is replayed. Once a mode enters the buffer, it is rehearsed as often as every other retained mode, so frequently sampled modes receive no additional replay.

Adding this loss to the fresh update from standard MaxRL (Tajwar et al., 2026) gives our primary method, Re:Max. This principle is modular and can be applied to any RLVR algorithm. For example, applying it to Dr.GRPO (Liu et al., 2025) yields Re:Dr. Solution modes discovered during training do not enter the on-policy reward group and contribute only through the separate replay loss. This separation allows one to vary the RLVR objective and the replay mechanism independently. Finally, our replay buffers contain only responses to training prompts; held-out evaluation therefore measures changes in the policy itself rather than retrieval or recitation from the buffer (App. D).

Fig. 7 illustrates how replay changes learning dynamics. In the standard RLVR setting, a policy learns only from modes present in the current sampling, which often leads previously discovered modes to disappear. Re:Max adds an additional incentive for the policy to continuously generate previously found solution modes that yield correct solutions. We note that this is fundamentally different from approaches that encourage diversity by modifying the on-policy learning signal to favor rare or diverse outcomes (Song et al., 2025; Li et al., 2026a), as we do not use mode identity to modify the task reward or advantage.

4 Experimental Design

In our main experiment, we examine whether replay improves correctness and solution-mode diversity across different RLVR objectives. We compare the impact of our replay component against both Dr.GRPO (Liu et al., 2025) and MaxRL (Tajwar et al., 2026). First, on all five ModeBench Level 1 train/test sets we carry out five seeds of post-training using Qwen2.5-0.5B, Falcon3-1B, and Qwen2.5-3B. We train for eight passes with $G = 16$ new rollouts. We note that replay does not increase this sampling budget. Instead, it performs an additional training update using at most $M = 16$ previously collected exemplars from the replay buffer for each visited prompt (Table 4). Second, we conduct the same experiment on ModeBench Levels 2 and 3 using only Qwen2.5-0.5B. Third, we compare our method with other approaches, including frequency-weighted replay, UCPO (Lochab et al., 2026), RLEP (Zhang et al., 2025), GAPO (Anschel et al., 2025), SetPO (Li et al., 2026a), and fixed Semantic-MaxEnt under a specific setting (Qwen2.5-0.5B, five seeds, Level 1).

We additionally evaluate 7 hosted models: GPT-5.6 Sol (OpenAI, 2026b), GPT-5.4 (OpenAI, 2026a), Grok 4.3 (xAI, 2026), Kimi K3 (Moonshot AI, 2026), DeepSeek V4 Pro (DeepSeek-AI, 2026), Claude

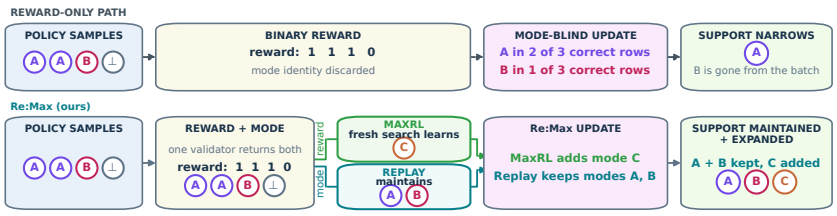

Figure 7: **Replay preserves successes that on-policy learning may stop seeing.** MaxRL learns from the policy's current samples, while replay stores one verified exemplar per discovered mode and revisits those exemplars with equal weight. Together, these two learning paths allow Re:Max to preserve previously discovered solution modes while continuing to explore for new ones.

Table 1: **Compared objectives under Qwen2.5-0.5B, ModeBench Level 1.** Each method is compared against the same task performed under Dr.GRPO; avg weights the five domains equally within seed (5-seeds). Bold marks a column's best. An em-dash means that there are few more-than-one solution pairs to evaluate, and a superscript gives the paired seed count where it falls below five. Red-to-green shading scales to each column's largest magnitude.

	Δ pass@8						Δ PCMD					
	Count				Pantry		Count				Pantry	
	Graph	down	Python	MathIR	Plan	Avg	Graph	down	Python	MathIR	Plan	Avg
Re:Dr *(ours)*	**+.646**	**+.191**	+.356	**+.285**	+.207	+.337	**+.559**	+.480	+.082	**+.017**	+.334	+.294
Re:Max *(ours)*	+.621	+.185	**+.509**	+.277	+.199	**+.358**	+.525	**+.504**	**+.249**	+.005	+.317	**+.320**
GAPO	+.175	+.011	+.009	−.054	**+.279**	+.084	+.131	+.340	.000	.000	**+.653**	+.225
SetPO	+.064	−.050	+.497	+.051	+.154	+.143	+.039	+.007	+.013	+.003	+.266	+.066
MaxRL (no replay)	+.213	+.101	.000	−.066	+.009	+.051	+.086	+.015	.000	.000	.000	+.020
RLEP-Dr	+.098	+.103	.000	+.126	.000	+.066	+.035	+.012	.000	−.001	.000	+.009
UCPO	−.003	+.133	.000	+.031	+.053	+.043	−.003 [2]	+.019	.000 [2]	−.001	.000 [2]	+.009
Semantic-MaxEnt	+.002	+.141	.000	−.121	+.201	+.045	+.004	+.020	.000	+.001	—	+.006
GRPO	+.037	+.159	.000	−.052	+.023	+.034	+.008	+.001	.000	+.001	.000	+.002
Untrained checkpoint	+.132	−.383	−.172	−.292	+.380	−.067	+.108	—	—	+.027	+.591	+.242

Opus 5 (Anthropic, 2026b), and Claude Opus 4.8 (Anthropic, 2026a) on ModeBench Levels 1–3, collecting eight responses per prompt and grading them through a fixed formatting normalizer so that presentation differences are not counted as verified alternatives (App. N). We carry out the same inference-only evaluation on Levels 1–5 across the available scales and benchmark levels of SmolLM2 (Ben Allal et al., 2025), Qwen2.5 (Qwen et al., 2024), Falcon3 (Falcon-LLM Team, 2024), and OLMo-2 (Team OLMo et al., 2024). These experiments are inference-only; no replay or training is applied.

5 Results

Our results are three-fold. First, we document solution-mode collapse under RLVR on ModeBench (Section 5.1). Second, we find that replay reverses this collapse without sacrificing correctness (Section 5.2), outperforming the 8 evaluated RLVR comparators on the five-domain aggregate. Third, we show that replay gains persist across difficulty levels (Section 5.3), KL anchoring, and domain changes.

5.1 RLVR concentrates solution modes

RLVR replaces diverse correct attempts with the same repeated answers. We compare each policy's solution-mode distribution before and after training using `PCMD`. Across Dr.GRPO, GRPO, and MaxRL results, we find fewer solution modes after RLVR post-training in 14 out of 15 domains for Qwen2.5-3B (Fig. 8), with the same pattern appearing across model scales and difficulty levels (Fig. 20). The starkest case is `Python`, where Qwen2.5-3B goes from solving 73 of 128 held-out prompts in multiple ways to as few as 22 after Dr.GRPO post-training.

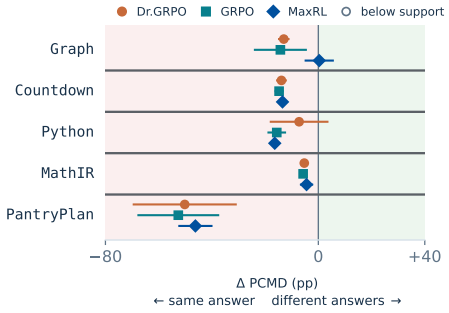

Figure 8: **Training reduces solution mode diversity.** We compare `PCMD` before and after RLVR on Qwen2.5-3B, over ModeBench Level 1 test set, under three objectives, five domains, five seeds. The figure shows the change in `PCMD` from before to after. Marks are means over training seeds with 95% intervals. In 14 of 15 comparisons, we find fewer solution modes after training. See Fig. 20 for other model scales.

As outputs concentrate, rollouts under RLVR lose training signal. Dr.GRPO and MaxRL learn from a rollout group only when it contains a mix of correct and incorrect responses. Thus, we audit every rollout group during training to track how often it provides a non-zero learning signal. We find that Re:Max systematically increases the percentage of rollout groups with a learning signal (App. F.4).

5.2 REPLAY IMPROVES BOTH ACCURACY AND DIVERSITY

Replay improves accuracy and diversity across RLVR algorithms and model scales. We add our proposed verified-mode replay mechanism to Dr.GRPO and MaxRL and compare each against its matched no-replay control across five MODEBENCH domains and three model scales. Across these comparisons, we find that Re:Max and Re:Dr (ours) shift the outcome toward both higher `pass@8` and greater solution-mode diversity (Fig. 9; Table 5).

Mode-diversity gains come from both retaining successes and balancing replay. To isolate the role of mode balancing, we compare uniform replay with a frequency-weighted variant that rehearses modes in proportion to their discovery rate. Holding replay buffer construction, replay advantage scheduling, and loss update unchanged, we find that uniform weighting preserves greater verified diversity, most clearly in `Graph`, `Countdown`, and `PantryPlan`. For example, the post-trained LM recovers roughly 0.6 additional modes per prompt in `Graph`, while also increasing `pass@8` by 6.6 percentage points. Across domains, these gains come without an aggregate difference in correctness (App. E.8, App. Fig. 19).

Alternative diversity mechanisms produce isolated rather than consistent gains. We compare Re:Max and Re:Dr (ours) against UCPO (Lochab et al., 2026), RLEP (Zhang et al., 2025), GAPO (Anschel et al., 2025), SetPO (Li et al., 2026a), and fixed Semantic-MaxEnt (Table 1, App. E.5). Some methods produce substantial gains in individual domains, e.g., GAPO reaches 2.9× its control's effective number of verified modes in `PantryPlan`. Yet gains are domain-specific, while Re:Max improves both metrics across domains, raising `pass@8` by 90% relative to Dr.GRPO with 1.9× its effective verified modes (Table 1; App. E).

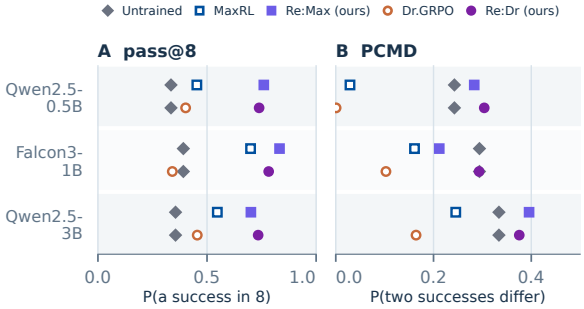

Figure 9: **Replay leads to higher accuracy and solution mode diversity.** Diamonds are initial LMs; open/filled markers are training without/with replay. Averages all five domains within seed. (App. Fig. 14 provides per-task results).

5.3 REPLAY REMAINS EFFECTIVE BEYOND THE BASE TRAINING SETTING



Figure 10: **Replay gains persist across difficulty levels.** Bars show the gain from adding replay to Dr.GRPO (Re:Dr) and MaxRL (Re:Max) at each MODEBENCH level, averaged across domains for Qwen2.5-0.5B. Replay improves both `PCMD` (top) and `pass@8` (bottom) at all three levels.

Replay's gains persist on harder task constructions. We repeat the replay-versus-control comparison within each of the first three MODEBENCH levels, holding the level and RLVR objective fixed. Re:Dr and Re:Max improve average `pass@8` and `PCMD` under Dr.GRPO and MaxRL, respectively, at every level (Fig. 10). At Level 2, for instance, `Graph` rises from 1.04 to 1.55 effective verified modes under Dr.GRPO, and from 1.13 to 1.48 under MaxRL. Gains vary across domains, but remain positive in aggregate even at Level 3 (App. E.3).

Replay leaves more usable alternative correct solutions when the task changes. We ask whether Re:Max and Re:Dr have a downstream impact on inference. Ideally, replay should make a model more able to give a correct answer when the original most-common answer is no longer available. To test this, we take a portfolio of eight generated answers, remove one option from the task, e.g. an ingredient in `PantryPlan` or a color in `Graph`, and re-check under the modified task. Across all 5 domains, Re:Max and Re:Dr are more likely to give a valid alternate answer. When no answer is valid, Re:Max and Re:Dr find a new valid answer in fewer additional attempts than RLVR or tuned-temperature sampling (App. L).

Reference KL and replay preserve different sources of diversity. We explore why Re:Max and Re:Dr gain solution diversity relative to the base model. A potential explanation is that the model retains the diversity of its base. We test this hypothesis by anchoring the policy to its base using KL divergence. We sweep the β-coefficient of a KL penalty to the initial policy and compare with Re:Max and Re:Dr. The anchor is limited by the diversity already present in its reference, whereas Re:Max and Re:Dr are limited by the modes their buffers discover. Averaged over the three domains with a reportable initial `PCMD`, matching diversity with a KL anchor costs model correctness, although in `Python` and `Pantry` the reference is already broad enough for the anchor to win (Fig. 11).

6 Related Work

Measuring solution diversity under alignment RLHF and reasoning RL can reduce output diversity, entropy, and high-budget coverage (Kirk et al., 2024; Dang et al., 2025; Cui et al., 2025; Yue et al., 2025). Semantic clustering infers it from language (Farquhar et al., 2024), which is general but depends on a judge. ModeBench instead defines solution identity through the same executable check that is used to assign reward. This gives us a narrower but cheaper measure of solution diversity, shared on training and evaluation. Preserving alternatives matters because sampling-based selection relies on generating multiple viable proposals (Brown et al., 2024; Snell et al., 2024; Cobbe et al., 2021; Lightman et al., 2024).

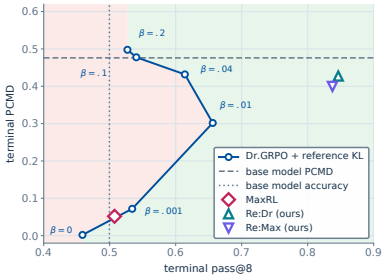

Figure 11: **Replay and KL anchoring preserve different diversity.** Means over the 3 domains with reportable initial `PCMD`, for Qwen2.5-0.5B. The KL anchor forms a curve because we sweep β-KL coefficients. Increasing β preserves more diversity, but at a lower `pass@8`. Green indicates higher `pass@8` than the anchor at the same diversity (App. E.7).

Expanding diversity. Outcome bonuses, set prediction, distribution matching, and uniform-correct objectives all reshape the learning signal (Song et al., 2025; Chen et al., 2025; Li et al., 2025; 2026a;b; Lochab et al., 2026), while MaxRL combines objectives across sampling budgets (Tajwar et al., 2026). Yet existing approaches still depend on what the current policy samples, where our analysis finds that solution diversity disappears. Replay can break this dependence on current samples. Standard replay methods (Lin, 1992; Rolnick et al., 2019; Zhang et al., 2025) revisit past successes in proportion to how often they occur. Instead, we homogeneously balance replay across solution modes, an idea related to quality-diversity methods (Mouret & Clune, 2015; Ecoffet et al., 2021).

7 Conclusion

We contribute ModeBench, a benchmark for studying solution mode diversity in RLVR, and Re:Max, a replay-based method that preserves discovered solution modes during RLVR. Across objectives, model scales, and task difficulties, we find that replay can retain solution diversity while improving correctness. Our results suggest that solution diversity may itself be a target for optimization. Future work can move beyond preserving discovered modes towards training models to produce desired distributions over ways to solve a problem.

Limitations. We acknowledge that replay retains only what sampling finds. We also recognize that mode count says nothing about the "worth" of additional solutions. Our experiments are designed such that the solution-equivalence relation is executable by construction, so future work is needed to evaluate the benefit and feasibility of replay in more open-ended solution spaces.

ACKNOWLEDGMENTS

Acknowledgments omitted for double-blind review.

AI USE STATEMENT

Generative AI assisted manuscript editing, mathematical analysis, and supporting code. The authors reviewed and verified the resulting content and take responsibility for the final manuscript.

ETHICS STATEMENT

The benchmark and experiment-specific data are synthetic; this study collects no human-subject or personal data. More verified modes are not inherently better: a distinct execution may be risky or costly even when it passes the validator, and `PCMD` does not measure downstream value. Systems that evaluate or optimize breadth should report correctness, invalidity, and application-specific costs alongside mode counts.

REPRODUCIBILITY STATEMENT

Every reported number is computed from a versioned result file. App. R documents evaluation provenance and links the released datasets, checkpoints, and analysis resources. The AI Use Statement above discloses generative-AI assistance.

REFERENCES

Marwa Abdulhai, Isadora White, Yanming Wan, Ibrahim Qureshi, Joel Leibo, Max Kleiman-Weiner, and Natasha Jaques. How LLMs distort our written language, 2026. URL https://arxiv.org/abs/2603.18161.

Barrett R. Anderson, Jash Hemant Shah, and Max Kreminski. Homogenization effects of large language models on human creative ideation. In *Proceedings of the 16th Conference on Creativity & Cognition*, pp. 413–425. Association for Computing Machinery, 2024. doi: 10.1145/3635636.3656204.

Oron Anschel, Alon Shoshan, Adam Botach, Shunit Haviv Hakimi, Asaf Gendler, Emanuel Ben Baruch, Nadav Bhonker, Igor Kviatkovsky, Manoj Aggarwal, and Gerard Medioni. Group-aware reinforcement learning for output diversity in large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 32394–32415, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1649.

Anthropic. Introducing Claude Opus 4.8, May 2026a. URL https://www.anthropic.com/news/claude-opus-4-8. Published May 28, 2026. Accessed September 22, 2026.

Anthropic. Claude Opus 5: Model documentation, 2026b. Hosted API model claude-opus-5, served through Anthropic Messages. Accessed September 22, 2026.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When smol goes big—data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

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, 2024. URL https://arxiv.org/abs/2407.21787.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@k training for adaptively balancing exploration and exploitation of large reasoning models, 2025. URL https://arxiv.org/abs/2508.10751.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models. arXiv preprint arXiv:2505.22617, 2025. URL https://arxiv.org/abs/2505.22617v1.

Xingyu Dang, Christina Baek, J. Zico Kolter, and Aditi Raghunathan. Assessing diversity collapse in reasoning. In *ICLR Workshop on Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL https://openreview.net/forum?id=AMiKsHLjQh.

DeepSeek-AI. DeepSeek-V4-Pro: Model documentation, 2026. Hosted API model DeepSeek-V4-Pro. Accessed September 22, 2026.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. RAFT: Reward rAnked FineTuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2304.06767.

Anil R. Doshi and Oliver P. Hauser. Generative AI enhances individual creativity but reduces the collective diversity of novel content. *Science Advances*, 10(28):eadn5290, 2024. doi: 10.1126/sciadv.adn5290.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590:580–586, 2021. doi: 10.1038/s41586-020-03157-9.

Falcon-LLM Team. The Falcon 3 family of open models, December 2024. URL https://falcon-lm.github.io/blog/falcon-3/. Published December 17, 2024. Accessed September 22, 2026.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, 2024. doi: 10.1038/s41586-024-07421-0.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

M. O. Hill. Diversity and evenness: A unifying notation and its consequences. *Ecology*, 54(2): 427–432, 1973.

Lou Jost. Entropy and diversity. *Oikos*, 113(2):363–375, 2006.

Yekyung Kim, Yapei Chang, Chau Minh Pham, and Mohit Iyyer. Argument collapse: LLMs flatten long-form public debate, 2026. URL https://arxiv.org/abs/2606.01736.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of RLHF on LLM generalisation and diversity. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PXD3FAVHJT.

Chenyi Li, Yuan Zhang, Bo Wang, Guoqing Ma, Wei Tang, Haoyang Huang, and Nan Duan. SetPO: Set-level policy optimization for diversity-preserving LLM reasoning, 2026a. URL https://arxiv.org/abs/2602.01062v1.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 110–119, 2016. doi: 10.18653/v1/N16-1014. URL https://aclanthology.org/N16-1014/.

Long Li, Zhijian Zhou, Jiaran Hao, Jason Klein Liu, Yanting Miao, Wei Pang, Xiaoyu Tan, Wei Chu, Zhe Wang, Shirui Pan, Chao Qu, and Yuan Qi. The choice of divergence: A neglected key to mitigating diversity collapse in reinforcement learning with verifiable reward, 2025. URL https://arxiv.org/abs/2509.07430v4.

Xiaozhe Li, Yang Li, Xinyu Fang, Shengyuan Ding, Peiji Li, Yongkang Chen, Yichuan Ma, Tianyi Lyu, Linyang Li, Dahua Lin, Qipeng Guo, Qingwen Liu, and Kai Chen. Beyond mode collapse: Distribution matching for diverse reasoning, 2026b. URL https://arxiv.org/abs/2605.19461.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992. doi: 10.1007/BF00992699.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2503.20783.

Anamika Lochab, Bolian Li, and Ruqi Zhang. Uniform-correct policy optimization: Breaking RLVR's indifference to diversity, 2026. URL https://arxiv.org/abs/2605.00365.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

Moonshot AI. Kimi K3: Open frontier intelligence, July 2026. URL https://www.kimi.ai/blog/kimi-k3. Official model announcement, July 16, 2026. Accessed September 22, 2026.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015. URL https://arxiv.org/abs/1504.04909.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, pp. 3878–3887. PMLR, 2018.

OpenAI. Introducing GPT-5.4, March 2026a. URL https://openai.com/index/introducing-gpt-5-4/. Published March 5, 2026. Accessed September 22, 2026.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, July 2026b. URL https://openai.com/index/gpt-5-6/. Published July 9, 2026. Accessed September 22, 2026.

Qwen, 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, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1811.11682.

E. H. Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.

Abhijeet Sinha, Sundari Elango, and Dianbo Liu. Expected return causes outcome-level mode collapse in reinforcement learning and how to fix it with inverse probability scaling, 2026. URL https://arxiv.org/abs/2601.21669.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Yuda Song, Julia Kempe, and Remi Munos. Outcome-based exploration for LLM reasoning, 2025. URL https://arxiv.org/abs/2509.06941.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning, 2026. URL https://arxiv.org/abs/2602.02710v3. Version 3.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious, 2024. URL https://arxiv.org/abs/2501.00656. First submitted December 31, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

xAI. Grok 4.3: Model documentation, 2026. URL https://docs.x.ai/developers/models/grok-4.3. API model documentation. Accessed September 22, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

Hongzhi Zhang, Jia Fu, Jingyuan Zhang, Kai Fu, Qi Wang, Fuzheng Zhang, and Guorui Zhou. RLEP: Reinforcement learning with experience replay for LLM reasoning, 2025. URL https://arxiv.org/abs/2507.07451.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 1097–1100, 2018. doi: 10.1145/3209978.3210080. URL https://arxiv.org/abs/1802.01886.

APPENDIX ORGANIZATION

The appendix is organized into eight parts, with sections and page numbers listed below.

Part I. Scope and measurement
A Metric and Objective Details . . . 17
A.1 Measuring success and diversity . . . 17
A.2 Reported endpoints and eligibility . . . 18
A.3 Why balance solution modes in replay . . . 18
Part II. Benchmark and method
B Benchmark and Evaluation Protocol . . . 19
B.1 Solution-mode definitions . . . 19
B.2 Dataset splits and validation . . . 19
B.3 `Python` execution and solution modes . . . 20
B.4 Matched level constructions . . . 20
B.5 Performance across benchmark levels . . . 21
B.6 Training and evaluation settings . . . 21
C Exact Domain Prompts . . . 23
C.1 `Graph` coloring . . . 23
C.2 `Countdown` . . . 23
C.3 `Python` factors . . . 23
C.4 `MathIR` action menu . . . 24
C.5 `PantryPlan` . . . 24
D Replay Algorithm . . . 24
D.1 Fresh task advantages . . . 24
D.2 Optimizer update . . . 25
D.3 Buffer construction and scheduling . . . 25
D.4 Matched control . . . 25
D.5 Verifier requirements and applicability . . . 26
D.6 Exemplar quality and length . . . 26
D.7 Semantic-MaxEnt implementation . . . 26
Part III. Training results
E Replay Effects Across Scales . . . 28
E.1 Terminal comparisons across scales . . . 28
E.2 Level 1 at Qwen2.5-7B . . . 29
E.3 Results on harder benchmark levels . . . 30
E.4 Diversity throughout training . . . 30
E.5 Supporting comparators . . . 30
E.6 Diversity-preserving comparators . . . 32
E.7 Reference KL tradeoffs . . . 33
E.8 Uniform versus fresh-frequency replay . . . 33
F Concentration and Replay Mechanisms . . . 33
F.1 Fresh 64-response evaluation . . . 33
F.2 Conditional-concentration robustness . . . 34
F.3 Diversity before and after RLVR-only training . . . 34
F.4 Sparse task gradients and output repetition . . . 35
F.5 Replay gains with active controls . . . 36
G Human-Authored Programming Evaluation . . . 36
Part IV. What diversity represents
H Prompt-Guidance Ablation . . . 38
I Inference-Time Decoding Controls . . . 38
J Sampling-Budget Ablation . . . 39
J.1 Sampling-budget design . . . 39
J.2 Discovery at larger sampling budgets . . . 39
J.3 Discovery at 512 draws . . . 39
K Inference-Time Alternatives and Mode Definitions . . . 40
K.1 Complementarity across hosted models . . . 40
K.2 Sensitivity to solution-mode identity . . . 40
L Portfolio Survival under Withdrawn Options . . . 40
L.1 Two withdrawals at once . . . 40
L.2 Decoding settings as a substitute . . . 41
L.3 Recovery after a withdrawal . . . 41
Part V. Hosted deployment analyses
M Inference-Only Concentration in GPT-5.6 Sol . . . 42
N Comparison across Hosted Deployments . . . 42
N.1 Hosted-model concentration . . . 42
N.2 `Python` prompt-interface sensitivity . . . 42
N.3 Reasoning-control comparison . . . 43
N.4 Temperature sensitivity across deployments . . . 44
N.5 Native provider outcomes . . . 44
Part VI. Mechanism and efficiency
O Mode Collapse and Verified Replay . . . 45
O.1 Solution-mode model and the known winner-take-all limit . . . 45
O.2 Rare modes receive vanishing on-policy signal . . . 46
O.3 Fixed-buffer replay preserves stored modes . . . 46
O.4 Scope of the idealized analysis . . . 47
P Compute and Storage Overhead . . . 47
P.1 Control design and compute matching . . . 47
P.2 Replay storage and runtime . . . 48
P.3 Longer-horizon control . . . 48
P.4 Wider-group control . . . 48

P.5 Single-slot replay 48
Part VII. Related-method comparison
Q Comparison with Diversity-Preserving RLVR Methods 50
Part VIII. Data and reproducibility
R Data and Reproducibility 51
R.1 Evaluation pairing 51
R.2 Released datasets and checkpoints 51
R.3 Analysis resources 51

Part I. Scope and measurement

A Metric and Objective Details

Correctness and verified-mode diversity are distinct properties of a policy. For a prompt x with reference specification s_x, the verifier maps a response to either failure or a verified solution mode:

$$V(\cdot, s_x) : \mathcal{Y} \to \mathcal{C}_x \cup \{\perp\}.$$

Here $\mathcal{C}_x$ is the set of valid solution modes and $\perp$ denotes failure. The learner observes a solution mode only for a response it generates; neither the full set $\mathcal{C}_x$ nor its size is available during training.

For $Y \sim \pi_\theta(\cdot \mid x)$, define the probability of a correct response and the conditional distribution over the solution mode of a verified correct response:

$$P_\theta^+(x) = \Pr[V(Y, s_x) \neq \perp], \qquad p_\theta^+(c \mid x) = \Pr[V(Y, s_x) = c \mid V(Y, s_x) \neq \perp].$$

We use the abbreviations P and q_c from the main text when the policy and prompt are clear. The conditional distribution is defined only when $P > 0$. If $P = 0$, `pass@`K, `distinct@`K, and `mean@`K are all zero, whereas `PCMD` is undefined.

We use *mode collapse* to describe increasing concentration over correct solution modes, particularly when correctness remains stable or improves. Transferring probability from a less probable correct mode to a more probable one decreases $\texttt{PCMD} = 1 - \sum_c q_c^2$. Accordingly, `PCMD` measures concentration among sampled correct modes rather than literal extinction of alternatives; finite evaluations characterize observed policy breadth.

Estimating `PCMD` requires two verified responses per prompt. We distinguish well-supported estimates with at least 30 eligible prompts from estimates based on fewer prompts; paired training comparisons require at least 20 eligible evaluation prompts in each arm (App. A.2).

A.1 Measuring success and diversity

For a prompt, let n_m be the number of verified responses assigned to solution mode m, and let $n_+ = \sum_m n_m$ be the total number of verified responses. When $n_+ \geq 2$, we estimate pairwise mode diversity as

$$\widehat{\texttt{PCMD}} = 1 - \frac{\sum_m \binom{n_m}{2}}{\binom{n_+}{2}} = 1 - \frac{\sum_m n_m(n_m - 1)}{n_+(n_+ - 1)}. \tag{4}$$

Thus `PCMD` is the probability that two independently sampled correct responses represent different solution modes. Under independent sampling from a fixed policy, this estimator is unbiased conditional on $n_+ \geq 2$.

We average prompt-level `PCMD` estimates with equal weight across eligible prompts rather than pooling response pairs across prompts. This prevents easier prompts, which produce more verified pairs, from receiving greater weight. Local evaluations generally use multiple response groups per prompt, while hosted evaluations use eight responses per prompt.

For K sampled responses, we additionally report

$$\texttt{pass@}K = \Pr(\text{at least one response is correct}),$$

$$\texttt{distinct@}K = \mathbb{E}[\text{number of distinct verified solution modes}],$$

and `mean@`K, the mean fraction of correct responses. Unlike `PCMD`, `distinct@`K depends on both correctness and how correct probability is distributed across solution modes.

`PCMD` is the Gini–Simpson diversity index applied to the conditional distribution over correct solution modes (Simpson, 1949). For interpretation, we also use the effective number of solution modes

$$N_{\text{eff}}^{(2)} = \frac{1}{1 - \texttt{PCMD}}, \tag{5}$$

the inverse Simpson index or Hill number of order two (Hill, 1973; Jost, 2006). A value of $N_{\text{eff}}^{(2)} = k$ corresponds to the diversity of a uniform distribution over k solution modes.

A.2 Reported endpoints and eligibility

We report pass@K for correctness, distinct@K for sampled solution breadth, and PCMD for diversity conditional on producing a correct response. Unlike distinct@K, PCMD depends only on the conditional distribution over correct solution modes and therefore separates solution diversity from overall correctness. A model can consequently achieve high pass@8 while repeatedly producing the same correct solution mode.

A prompt-level PCMD estimate requires at least two verified responses. For evaluation summaries, filled markers indicate at least 30 eligible prompts, hollow markers indicate 5–29, and estimates based on fewer than five prompts are omitted. Missing PCMD values therefore indicate insufficient verified responses rather than zero diversity.

For paired training comparisons, both arms must contain at least 20 eligible prompts. Unless otherwise stated, each arm is averaged over its own eligible prompt set, so differences can reflect both the conditional solution distribution and which prompts satisfy the eligibility criterion.

We also report distinct@K $-$ pass@K as the expected number of verified solution modes beyond the first. This quantity requires no knowledge of the full set of valid solution modes but, unlike PCMD, remains coupled to correctness.

A.3 Why balance solution modes in replay

Replay gives each stored solution mode equal weight, regardless of how often that mode was discovered. A simple probability decomposition clarifies the effect of this choice. For a buffer $\mathcal{B}_x$ containing k solution modes, let $p(m)$ denote the probability of mode m, let $P_{\mathcal{B}}$ be their total probability mass, and let U_k be uniform over the buffered modes. Then

$$-\frac{1}{k}\sum_{m\in\mathcal{B}_x}\log p(m) = -\log P_{\mathcal{B}} + \log k + \mathrm{KL}(U_k \,\|\, p(m \mid m \in \mathcal{B}_x))\,. \tag{6}$$

The decomposition therefore shows that the replay objective favors both greater total probability on buffered solution modes and a more balanced distribution among them. At fixed buffered probability mass, balancing the modes increases the expected number of distinct buffered modes observed in repeated samples, with the maximum attained by the uniform allocation. This provides the motivation for weighting discovered solution modes equally rather than in proportion to how frequently fresh rollouts rediscover them.

The implemented replay loss operates on one stored exemplar per solution mode, so the decomposition gives the mode-level motivation for the training objective. The fresh-rollout frequency ablation in App. E.8 then tests this balancing principle directly in neural training.

Part II. Benchmark and method

B BENCHMARK AND EVALUATION PROTOCOL

MODEBENCH uses executable verifiers to determine both correctness and solution-mode identity. Invalid responses receive reward zero, while verified responses receive reward one and are assigned to a solution mode. The learner observes only modes that it generates and is never given the complete set or number of valid modes. These quantities are not used to schedule replay or determine when training stops. The benchmark is available in the public release, and the training experiments in this paper use Levels 1–3.

Table 2: **Executable solution modes in the five MODEBENCH domains.** The verifier determines both correctness and solution-mode identity. *Certified modes* gives the mean and range of the Level-1 evaluation catalogue; these catalogues need not enumerate every accepted solution.

Domain	Generated answer / check	Solution mode	Equivalent responses	Certified modes
`Graph` coloring	Complete assignment; check fixed colors and edges	Color vector	Spacing only	6.27 4–12
`Countdown`	Parse operands and execute exactly	Normalized AST	Commutative order	4.52 2–8
`Python` factors	Execute restricted lambda externally	Return vector	Programs with the same return vector	229.44 16–1,512
`MathIR`	Execute bounded action program	State trajectory	Programs with the same states	5.00 5
`PantryPlan`	Check ingredient allocation and all constraints	Ingredient support	Quantity and ordering aliases	18.19 8–40

B.1 SOLUTION-MODE DEFINITIONS

The five domains instantiate different notions of a distinct correct solution. In `Countdown` and `MathIR`, solution modes correspond to different executable routes to an answer. In `Python`, they correspond to different return vectors on the fixed inputs specified by the prompt. In `Graph` and `PantryPlan`, they correspond to different valid answers. These definitions identify verified solution behavior rather than latent reasoning processes.

Sensitivity to the mode definition. The chosen equivalence relation determines which verified responses count as the same solution mode. App. K.2 therefore repeats the analysis under coarser task-relevant definitions. Most replay effects persist, although `Python` is more sensitive to the equivalence relation than the other domains. Training always uses the benchmark's original solution-mode definitions; the coarser analysis is used only as a robustness check.

Executable-answer setting. Training prompts request executable answers rather than unrestricted reasoning traces, allowing solution-mode identity to be determined without a learned judge or embedding-similarity threshold. The experiments therefore measure diversity among executable solutions, not diversity in latent reasoning or unrestricted chain-of-thought. Evaluating those broader notions would require a separate definition and validation of solution identity.

B.2 DATASET SPLITS AND VALIDATION

Each domain uses disjoint training and evaluation splits (Table 4), with additional development splits for `PantryPlan` and Level 2 (App. B.4). `Python` and `MathIR` each contain 384 training and 128 evaluation problems. Every `Python` problem is validated to admit at least two distinct correct return vectors, and every certified `MathIR` trajectory is checked using the same interpreter applied to model outputs.

PantryPlan. PantryPlan contains 384 training, 64 development, and 128 evaluation prompts across four problem families. Evaluation prompts admit a mean of 18.19 feasible ingredient supports, ranging from 8 to 40. The verifier uses exact arithmetic to check inventory, serving increments, mass, nutrient bounds, and dietary restrictions, and defines the solution mode by the set of ingredients used. These checks establish satisfaction of the benchmark constraints rather than taste, preparation feasibility, or food safety.

B.3 PYTHON EXECUTION AND SOLUTION MODES

The Python verifier accepts restricted expressions of the form `lambda n: EXPR`, limited to 240 characters and 64 AST nodes. Expressions may use integer arithmetic, comparisons, Boolean operators, and conditionals, but may not use calls, imports, attributes, comprehensions, containers, assignments, loops, or identifiers other than `n`. Level-1 prompts contain four fixed inputs, and every problem admits at least two valid solution modes.

External execution. Candidate expressions execute in an isolated Python process with empty `__builtins__` and fixed time limits. Malformed expressions, execution failures, timeouts, non-integer outputs, and failed divisor checks receive zero reward.

Solution-mode identity. For fixed inputs $n_1, \ldots, n_J$, a verified response produces integers $d_1, \ldots, d_J$ satisfying

$$1 < d_j < n_j, \qquad d_j \mid n_j \qquad \text{for all } j.$$

Its solution mode is the ordered return vector $(d_1, \ldots, d_J)$, so two programs represent the same mode exactly when they return the same vector on all benchmark inputs. The exact number of valid return vectors is used only to validate the dataset and is never exposed to the learner.

B.4 MATCHED LEVEL CONSTRUCTIONS

Levels 2–5 increase task complexity while preserving the verifier and solution-mode definition within each domain. Level 2 supports the harder-task training experiments, while Levels 3–5 support scale-calibrated evaluations.

Level 2. Level 2 contains 384 training, 128 development, and 128 evaluation problems, with no shared problem identities across splits or with Level 1. Its training-set distribution of certified solution-mode counts is matched to the Level-1 construction reference. The evaluation populations are not identically distributed across levels in every domain, so cross-level differences should not be interpreted as the isolated causal effect of difficulty. Within each level, replay and control conditions use the same prompts, seeds, fresh objective, and training budget.

Table 3: **Level 2 increases task complexity while preserving solution-mode identity.** The verifier and definition of a solution mode are unchanged from Level 1.

Domain	Level 1	Level 2
Graph coloring	Three hidden vertices	Four hidden vertices
Countdown	Three operands	Four operands, with values up to 14
Python factors	Four executed cases, all at most 96	Four cases, including a value above 96
MathIR	One-sided linear families	Variable-on-both-sides families
PantryPlan	Base dietary and composition constraints	More active exclusions and tighter composition constraints

Scale-calibrated Levels 3–5. Levels 3–5 are constructed so that progressively larger fixed models face approximately the same correctness difficulty as Qwen2.5-0.5B on Level 1. Level 3 uses Qwen2.5-3B, Level 4 uses Qwen2.5-7B, and Level 5 uses Qwen2.5-14B. Construction parameters are selected on development problems using `pass@1` and `pass@8`, with tolerances of .04 and .08 from the Level-1 reference; solution diversity is not part of the calibration objective. Level 5 satisfies these criteria in all five domains, while Level 4 satisfies them in all domains except MathIR. We therefore include Level-4 MathIR where relevant but do not treat it as difficulty-matched to the Level-1 reference.

B.5 PERFORMANCE ACROSS BENCHMARK LEVELS

We evaluate untrained checkpoints across all five benchmark levels to characterize how correctness and solution diversity vary with task construction before RLVR training. Each local-model evaluation uses 128 held-out prompts per domain and level, with four independent groups of eight responses per prompt. For `pass@8` and `distinct@8`, we average the four groups within each prompt, while `PCMD` is estimated from the pooled responses.

Higher levels generally reduce correctness, although the relationship is not strictly monotone for every model and domain. Solution diversity does not vary in lockstep with correctness or model scale, so benchmark difficulty cannot be inferred from `PCMD` alone. These measurements describe the benchmark populations and are separate from the matched training comparisons, which always compare replay and control under the same evaluation conditions.

B.6 TRAINING AND EVALUATION SETTINGS

Table 4: **Training and evaluation settings for the matched replay comparisons.** Replay and control use the same initialization, fresh-rollout budget, optimizer updates, prompts, and evaluation procedure.

Component	Setting
Data	384 training and 128 disjoint evaluation prompts per domain and level; evaluation prompts are never used for training or replay.
Fresh rollouts	$G = 16$, temperature 1, top-$p = 1$, with one prompt group per optimizer update.
Training horizon	Eight passes over the training set, totaling 3,072 optimizer updates, with one PPO epoch per batch.
Evaluation	Four groups of eight responses per prompt at temperature 1 and top-$p = 1$ at passes $0, .5, \ldots, 8$; final comparisons use the last checkpoint.
Replay	At most $M = 16$ solution modes per prompt, one stored exemplar per mode, deterministic round-robin buffer selection, and $\lambda_{\text{replay}} = .10$.
Training seeds	Five seeds per model–domain condition, with paired comparisons using common seeds whenever available.
Learning rate	Qwen2.5-0.5B and Falcon3-1B use a constant 2×10^{-7} learning rate; Qwen2.5-3B uses cosine decay from 10^{-7} to 10^{-8} with 10% warmup.
Optimizer	Adam in `bf16` with $\beta_1 = .9$, $\beta_2 = .95$, zero weight decay, gradient-norm clipping at 1.0, and gradient checkpointing.
Policy update	Parameters are synchronized after each update, each rollout is used once, and the symmetric PPO clipping threshold is $\varepsilon = .2$.
Response tokens	`Graph`/`Countdown`: 192; `Python`: 192/512/192 at 0.5B/1B/3B; `MathIR`: 64/128/64; `PantryPlan`: 8.

Replay and control additionally execute the same validation, buffer construction, scheduling, and exemplar-scoring code, with the replay gradient set to zero in the control. They therefore match optimizer updates and fresh-rollout budgets but not total computation or wall-clock time (App. P.1).

For each seed, `pass@8` and `distinct@8` are averaged over four eight-response groups for each of the 128 evaluation prompts. Evaluation responses are excluded from training and replay and do not affect checkpoint selection or stopping.

Final-checkpoint comparisons use paired training seeds. When all five paired seeds are available, we report nominal 95% Student-t intervals; comparisons with fewer seeds report their sample size and mean. Diversity comparisons additionally apply the eligibility criteria stated with each analysis.

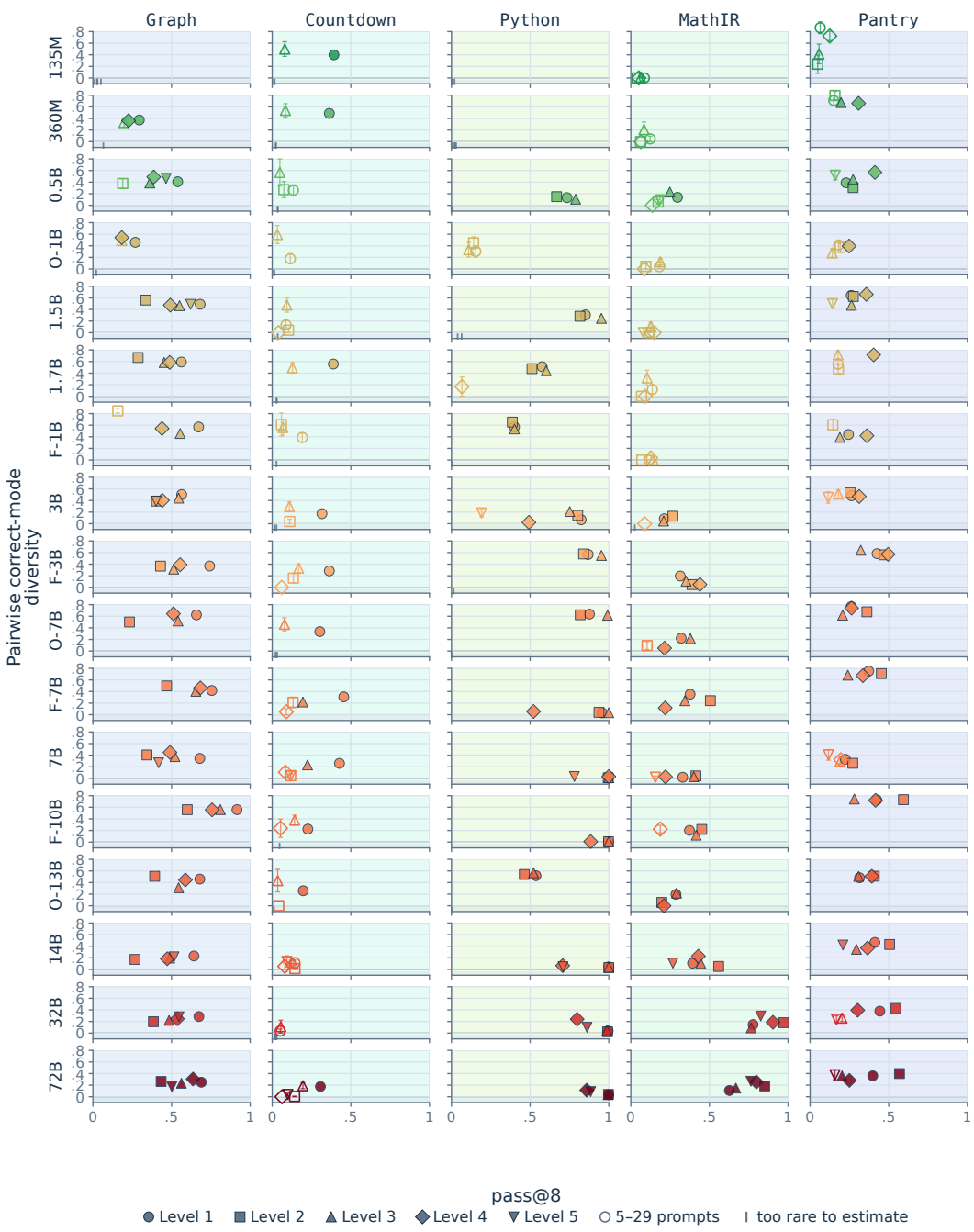

Figure 12: **Correctness and conditional solution diversity vary separately across benchmark levels.** Each point is an untrained model evaluated on one domain and level, with `pass@8` on the horizontal axis and `PCMD` on the vertical axis. Filled markers indicate sufficient verified responses for the primary diversity estimate, while hollow markers indicate limited support.

C EXACT DOMAIN PROMPTS

Each domain's training prompt contains one system message and one user message, without few-shot examples. Training and evaluation use the same template within each method. The examples below reproduce the first prompt from each Level-1 evaluation split in Qwen's chat format. Falcon3 uses the same message text with its own <|system|> and <|user|> markers. Models evaluated before benchmark training use their tokenizer's native chat template (App. B.5). App. B.4 describes the different problems and prompting conditions at Levels 2–5.

The examples preserve the prompt text. A continuation line indented by two spaces indicates a display line break; replacing that break and indentation with one space recovers the original text. In the `PantryPlan` example, a line break without this indentation is part of the prompt.

C.1 GRAPH COLORING

384 training and 128 evaluation prompts.

```
<|im_start|>system
Return only the final answer inside \boxed{}. Do not explain.<|im_end|>
<|im_start|>user
Color this graph with vertices 1 through 6. The undirected edges are: (1,2), (1,3),
  (3,5), (3,6), (4,6), (5,6). Assign each vertex one color from {1, 2, 3} so that no
  edge has the same color on both endpoints. A partial coloring is 2??1?2, where ?
  means uncolored. Keep every shown digit fixed. There are exactly 3 question marks.
  Return exactly one final answer and no explanation: exactly 3 digits, each one 1,
  2, or 3, for the missing positions from left to right, inside \boxed{}.<|im_end|>
<|im_start|>assistant
```

C.2 COUNTDOWN

384 training and 128 evaluation prompts.

```
<|im_start|>system
Return only the final answer inside \boxed{}. Do not explain.<|im_end|>
<|im_start|>user
Using the numbers [3, 6, 9], create an arithmetic expression that equals 18. Use
  each given number exactly once. You may use +, -, *, /, and parentheses. Return
  exactly one final answer and no explanation: the expression inside
  \boxed{}.<|im_end|>
<|im_start|>assistant
```

C.3 PYTHON FACTORS

384 training and 128 evaluation prompts.

```
<|im_start|>system
Return only the final answer inside \boxed{}. Do not explain.<|im_end|>
<|im_start|>user
Write one pure Python function with the exact form lambda n: EXPR. An external
  Python tool will call it once for each n in [18, 82, 91, 93]. For every call,
  return an integer d satisfying 1 < d < n and n % d == 0. Different valid return
  vectors are different solution modes. You may use integer literals, n, +, -, *,
  //, %, comparisons, Boolean operators, and conditional expressions; calls,
  imports, attributes, containers, and other names are forbidden. Return exactly the
  one-line lambda inside \boxed{} and no explanation.<|im_end|>
<|im_start|>assistant
```

C.4 MATHIR ACTION MENU

384 training and 128 evaluation prompts.

```
<|im_start|>system
Return only the final answer inside \boxed{}. Do not explain.<|im_end|>
<|im_start|>user
MathIR linear-menu-v1. Solve for x by choosing an executable action sequence. Every
  selected action is applied exactly to BOTH sides of the current equation, followed
  by exact simplification. Bindings: a=2, b=-9, c=-1. Initial equation: x/a + b = c.
  Action menu: A means div(a); B means add(b); C means sub(b); D means sub(c); E
  means sub(mul(a,b)); F means mul(a). Choose 1-4 action IDs. Illegal operations,
  repeated states, and paths that do not finish with x isolated are rejected. Return
  only the IDs separated by semicolons inside the answer box; for example formatting
  only: \boxed{A;C}. Do not return x's numeric value.<|im_end|>
<|im_start|>assistant
```

C.5 PANTRYPLAN

384 training and 128 evaluation prompts.

```
<|im_start|>system
Return only six binary digits and no other text. Each digit maps to the
  corresponding Pantry row in printed order. A trusted environment will choose exact
  quantities on precisely the selected rows.<|im_end|>
<|im_start|>user
Create one feasible high fiber snack from this frozen pantry.
Nutrition values are per 100 g. Quantities are exact integer grams.

Pantry:
- almonds: available=100g, step=25g, minimum-if-used=50g; energy=584kcal,
  protein=21.5g, fiber=10.8g, sodium=0.0mg; tags=tree_nut
- banana: available=125g, step=25g, minimum-if-used=50g; energy=97.0kcal,
  protein=0.74g, fiber=4.62g, sodium=0.0mg; tags=none
- carrots: available=150g, step=25g, minimum-if-used=50g; energy=45.0kcal,
  protein=0.941g, fiber=3.1g, sodium=86.6mg; tags=none
- grape_tomatoes: available=100g, step=25g, minimum-if-used=50g; energy=27.0kcal,
  protein=0.83g, fiber=2.1g, sodium=6.0mg; tags=none
- navel_orange: available=150g, step=25g, minimum-if-used=50g; energy=47.0kcal,
  protein=0.91g, fiber=2.0g, sodium=9.0mg; tags=none
- sunflower_seeds: available=125g, step=25g, minimum-if-used=50g; energy=571kcal,
  protein=18.9g, fiber=7.22g, sodium=0.0mg; tags=none

Use 2 to 4 ingredients.
Forbidden tags: none
Targets:
- mass_g: minimum 125, maximum 200
- energy_kcal: minimum 239.4, maximum 432.25
- protein_g: minimum 6.423
- fiber_g: minimum 3.478
- sodium_mg: maximum 37.6

Choose the ingredient support as a six-bit mask in the exact Pantry row order. Bit 1
  includes that row and bit 0 excludes it.<|im_end|>
<|im_start|>assistant
```

D REPLAY ALGORITHM

Verified replay augments learning from fresh rollouts with a likelihood loss on stored verified solutions. Each prompt x has a buffer $\mathcal{B}_x$ containing at most M exemplars, with one exemplar per discovered solution mode. At each update, a round-robin schedule selects one nonempty buffer and gives each stored solution mode equal weight in the replay loss. The fresh objective determines whether the resulting method is Re:Dr or Re:Max; the replay mechanism is otherwise identical.

D.1 FRESH TASK ADVANTAGES

For a group of G fresh responses with binary rewards R_i and $R = \sum_i R_i$, Dr.GRPO uses

$$A_i^{\mathrm{Dr}} = R_i - \frac{R}{G},$$

while MaxRL uses

$$A_i^{\mathrm{MaxRL}} = \begin{cases} GR_i/R - 1, & R > 0, \\ 0, & R = 0. \end{cases}$$

In a mixed group, MaxRL assigns advantage -1 to failures and gives each successful response greater weight when successes are scarce (Tajwar et al., 2026). Both objectives assign zero fresh-task advantage when every response in the group receives the same reward. Stored exemplars do not receive these fresh advantages and instead contribute through the replay loss defined below.

D.2 Optimizer update

Algorithm 1 Verified replay update. The fresh objective determines Re:Dr or Re:Max.

Require: Policy π_θ; prompt x and verifier specification s_x; group size G; fresh objective $o \in \{\mathrm{Dr.GRPO}, \mathrm{MaxRL}\}$; replay weight $\lambda_{\mathrm{replay}}$; buffer capacity M; buffers $\mathcal{B}$; round-robin cursor σ
Ensure: Updated policy, buffers, and cursor
Draw the on-policy group $\mathcal{G} = \{y_i \sim \pi_\theta(\cdot \mid x)\}_{i=1}^{G}$
for $i = 1, \ldots, G$ **do**
  Assign solution mode $m_i \leftarrow V(y_i, s_x)$
  Set $R_i \leftarrow \mathbf{1}\{m_i \neq \perp\}$
end for
Compute fresh task advantages A_i^o from $\{R_i\}_{i=1}^{G}$
After validating the full group, retain one exemplar for each newly discovered solution mode $m_i \neq \perp$ in $\mathcal{B}_x$, subject to capacity M
if at least one replay buffer is nonempty **then**
  $(z, \sigma') \leftarrow \textsc{NextBuffer}(\mathcal{B}, \sigma)$ using the fixed round-robin schedule
  For each stored exemplar $e_m \in \mathcal{B}_z$, compute $\ell_m = |e_m|^{-1} \log \pi_\theta(e_m \mid z)$
  $L_{\mathrm{replay}} \leftarrow -|\mathcal{B}_z|^{-1} \sum_{m \in \mathcal{B}_z} \ell_m$
else
  $L_{\mathrm{replay}} \leftarrow 0$
  $\sigma' \leftarrow \sigma$
end if
$\theta' \leftarrow \textsc{Update}\big(\theta, L_{\mathrm{PPO}}(A^o) + \lambda_{\mathrm{replay}} \frac{G-1}{G^2} L_{\mathrm{replay}}\big)$

D.3 Buffer construction and scheduling

Each training prompt has its own replay buffer. A verified response is eligible for storage once the verifier assigns it a valid solution mode. The full rollout group is validated before the buffer is updated, and the buffer stores at most one exemplar for each discovered solution mode. When multiple responses first instantiate the same mode, we retain the lexicographically smallest normalized response-token sequence, making exemplar selection independent of rollout order. Each buffer contains at most $M = 16$ solution modes, with no eviction; modes discovered after the buffer is full receive no replay support.

At each update, a deterministic round-robin schedule selects one nonempty buffer. The policy scores every stored exemplar in that buffer by teacher forcing on its original prompt, and each exemplar contributes its mean response-token log probability excluding prompt and padding tokens. The replay loss is the negative mean of these exemplar scores, weighted by $\lambda_{\mathrm{replay}}(G-1)/G^2$ (Algorithm 1). The factor $(G-1)/G^2$ matches the contribution of a single successful response in a centered Dr.GRPO group of size G.

Replay therefore uses only solution modes discovered during training and does not require the full set of valid solutions or any evaluation outputs.

D.4 Matched control

Replay and control use the same validation, buffer construction, scheduling, exemplar scoring, optimizer updates, and fresh-rollout budget. The control executes the same replay machinery but sets the replay gradient to zero, so the replay learning signal is the only difference in the objective.

As training progresses, the policies can generate different responses and therefore accumulate different buffers and replay workloads. App. P.1 quantifies this overhead and compares replay against controls with larger fresh-rollout budgets.

D.5 VERIFIER REQUIREMENTS AND APPLICABILITY

Replay requires a deterministic rule for assigning each verified response to a solution mode. This identity must be computable during training at reasonable cost and must reflect the task's intended notion of distinct solutions. In MODEBENCH, the executable verifier determines both correctness and solution-mode identity, so buffer construction requires neither a learned judge nor an embedding-similarity threshold.

Verified replay applies whenever training-time verification can assign correct responses to task-relevant solution modes. Final-answer-only graders collapse distinct derivations of the same answer into one mode, so preserving derivational alternatives requires a finer identity rule. Consistent with this distinction, the single-slot replay control retains part, but not all, of Re:Dr's diversity benefit (App. P.5).

`MathIR` demonstrates a path-based identity by defining modes through executed state trajectories. More open-ended tasks could analogously use a validated learned judge or another structured equivalence rule. Executable identities in MODEBENCH provide a deterministic reference setting for studying this mechanism before extending it to learned mode definitions.

D.6 EXEMPLAR QUALITY AND LENGTH

Verification establishes that each retained exemplar satisfies the task constraints and assigns it to a solution mode. Replay then gives discovered modes equal buffer representation without introducing a separate learned ranking over exemplar quality.

Each solution mode occupies one buffer entry, and its exemplar contributes its mean response-token log probability,

$$\ell_m = |e_m|^{-1} \log \pi_\theta(e_m \mid z).$$

Thus longer exemplars do not receive additional buffer entries or loss weight proportional to their token count. Mean-token normalization gives each stored exemplar equal weight in the replay loss, but it does not imply that the trained policy assigns equal probability to the corresponding solution modes.

We apply no additional response-length or quality filters beyond the generation limits and executable verifier, keeping the replay target aligned with the benchmark's verified notion of a valid alternative.

D.7 SEMANTIC-MAXENT IMPLEMENTATION

Fixed Semantic-MaxEnt adds a frequency-based bonus to fresh verified responses. For each prompt, it tracks solution modes observed in previous successful rollouts and among the other successful responses in the current group. Let $n_x(a)$ count previous successful responses with solution mode a, and let $m_{-i}(a)$ count successful responses with that mode among the other members of the current group. Define

$$N_x = \sum_a n_x(a), \qquad S_{-i} = \sum_a m_{-i}(a),$$

and let $\mathcal{K}_i$ contain the solution modes observed in either source. We estimate

$$\hat{p}_i(a) = \frac{n_x(a) + m_{-i}(a) + 1}{N_x + S_{-i} + |\mathcal{K}_i| + 1}, \qquad \hat{p}_i(\text{UNSEEN}) = \frac{1}{N_x + S_{-i} + |\mathcal{K}_i| + 1}. \tag{7}$$

A newly observed solution mode receives the probability assigned to the UNSEEN category.

For a successful response with solution mode a_i, the added advantage is

$$A_i^{\text{sem}} = \eta \frac{\min\{-\log \hat{p}_i(a_i), \kappa\} - \mathbb{E}_{\hat{p}_i}[\min\{-\log \hat{p}_i(a), \kappa\}]}{\kappa}, \qquad \eta = .10, \quad \kappa = 5. \tag{8}$$

The same bonus is applied at every token of the response, while unsuccessful responses receive no bonus. Counts from the current group are added only after scoring, so a response does not affect its own bonus.

This mechanism favors underrepresented verified solution modes only when they appear in fresh rollouts. Unlike replay, it does not optimize the likelihood of stored exemplars whose solution

modes are absent from the current group. Because $\hat{p}_i$ is a frequency-based predictor rather than the model's current conditional distribution over correct solution modes, Semantic-MaxEnt serves as a fresh-rollout frequency baseline rather than the same objective as stored-mode replay.

Part III. Training results

E Replay Effects Across Scales

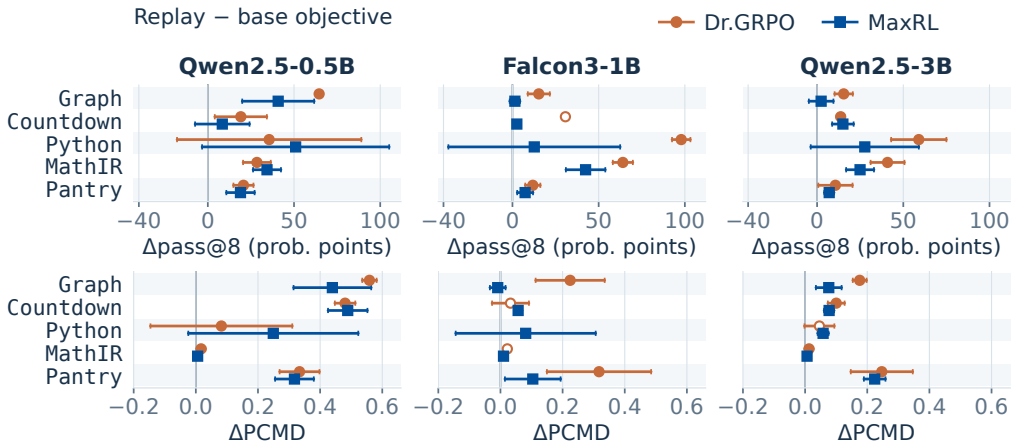

Figure 13: **Replay gains persist across model scales.** Points show terminal Re:Dr minus Dr.GRPO and Re:Max minus MaxRL effects on `pass@8` and `PCMD` across three model scales and five domains. Filled markers use five paired seeds, hollow markers use fewer, and bars show nominal 95% paired intervals where available.

E.1 Terminal comparisons across scales

Replay and control conditions share initialization, prompts, optimizer settings, training updates, fresh-rollout budget, verifier, and evaluation requests. The control executes the same replay machinery with zero replay gradient, although total compute is not matched because replay processes stored exemplars.

Across model scales, replay improves both correctness and conditional solution diversity on average. Table 5 summarizes the paired cross-domain effects at the terminal checkpoint.

Domain-level effects. Re:Dr increases mean `PCMD` over Dr.GRPO in all 14 measurable model–domain comparisons, while Re:Max exceeds MaxRL in 14 of 15. The domain-level pattern is therefore broad across models and objectives. Effect sizes vary, with Falcon3-1B `Graph` providing the lone near-zero Re:Max diversity comparison and Qwen2.5-3B showing positive Re:Max effects in all five domains.

Each paired `PCMD` contrast uses prompts with sufficient verified responses in both arms, so eligibility is determined within the corresponding model–domain comparison. The reported effects therefore compare matched conditional solution distributions within each evaluable cell.

Table 5: **Replay improves mean correctness and conditional diversity across model scales.** Entries are replay minus the matched non-replay objective after eight training passes. `pass@8` averages all five domains within matched seeds. `PCMD` averages a fixed set of domains with defined paired estimates for all contributing seeds. Falcon3-1B Re:Dr `pass@8` uses four common seeds; every other entry uses five.

Model	Comparison	Δpass@8	ΔPCMD
Qwen2.5-0.5B	Re:Dr − Dr.GRPO	+.337	+.294
Falcon3-1B	Re:Dr − Dr.GRPO	+.442	+.271
Qwen2.5-3B	Re:Dr − Dr.GRPO	+.279	+.134
Qwen2.5-0.5B	Re:Max − MaxRL	+.307	+.300
Falcon3-1B	Re:Max − MaxRL	+.133	+.049
Qwen2.5-3B	Re:Max − MaxRL	+.154	+.088



Figure 14: **Replay gains vary by domain and model scale.** Final `pass@8` and `PCMD` across three model scales for Dr.GRPO, MaxRL, and their replay variants. Markers show seed means; estimates with insufficient verified responses are omitted.

Cross-domain summary. Positive mean replay effects appear under both fresh objectives at all three model scales. The domain-level results in Fig. 14 show that the magnitude of these gains depends on the task and fresh objective.

E.2 Level 1 at Qwen2.5-7B

We additionally run one matched MaxRL–Re:Max seed on Qwen2.5-7B for `Graph`, `Countdown`, and `MathIR`. These runs use the same training horizon, prompts, verifier, evaluation procedure, and replay schedule as the smaller-scale comparisons, with one matched seed per domain.

Across the three domains, mean `pass@8` increases from .342 for MaxRL to .789 for Re:Max. On `Graph` and `Countdown`, where both arms have measurable conditional diversity, mean `PCMD` increases from .028 to .245. On `MathIR`, MaxRL produces no verified response at the final checkpoint, whereas Re:Max reaches `pass@8` .744 and `PCMD` .017. These results extend the replay pattern to Qwen2.5-7B; the one-seed design is used as a scale extension rather than for estimating a model-size trend.

E.3 Results on harder benchmark levels

We evaluate the four primary methods on Level 2 using Qwen2.5-0.5B. Correctness comparisons use five matched seeds, while PCMD is reported only when both arms contain enough prompts with multiple verified responses.

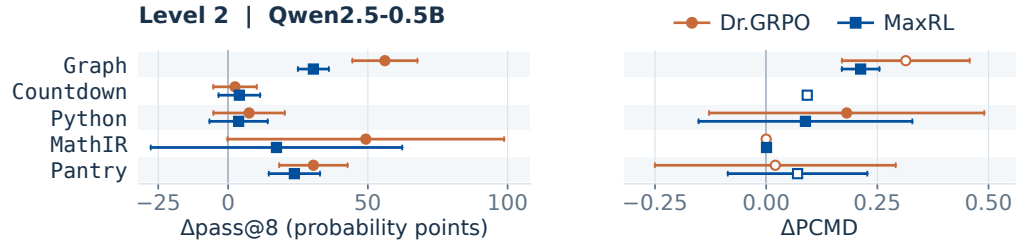

Figure 15: **Replay improves Level-2 `Graph` accuracy and conditional solution diversity.** Points show terminal Re:Dr minus Dr.GRPO and Re:Max minus MaxRL effects on pass@8 and PCMD. Bars show nominal 95% paired intervals where at least two eligible seed pairs are available.

Level-2 `Graph`. Replay improves both correctness and conditional solution diversity substantially on Graph. After eight training passes, Dr.GRPO and MaxRL reach pass@8 of .202 and .434, compared with .763 and .739 for Re:Dr and Re:Max. Among the eligible paired seeds, PCMD increases from .039 to .353 for Re:Dr versus Dr.GRPO and from .112 to .325 for Re:Max versus MaxRL. The corresponding paired PCMD gains are $+.314$ and $+.212$.

Beyond Graph, replay also improves PantryPlan correctness under both objectives and yields positive Python diversity point estimates, while MathIR remains concentrated under all four methods. Level 2 therefore shows that replay gains persist on harder constructions, with effect size varying by domain. Because the Level-2 task populations and some prompt guidance differ from Level 1, cross-level contrasts summarize benchmark regimes rather than isolate difficulty alone.

E.4 Diversity throughout training

We examine PCMD throughout training to distinguish terminal differences from transient variation. Conditional-diversity estimates are reported only when enough held-out prompts contain at least two verified responses, so the contributing prompts and seeds can vary across checkpoints.

Across scales, the largest separation appears in PantryPlan, where the no-replay objectives approach zero terminal diversity at the smaller scales while replay retains multiple correct solution modes. The trajectories show that the terminal replay advantage reflects sustained differences during training rather than only a final-checkpoint fluctuation.

E.5 Supporting comparators

GRPO changes the within-group reward normalization, UCPO redistributes fresh on-policy advantage, and sparse RLEP-Dr replays verified successes without balancing them by solution mode. We evaluate GRPO across three model scales, while UCPO and RLEP-Dr cover the two smaller scales. Fixed Semantic-MaxEnt adds a frequency-based bonus to fresh verified responses (App. D.7).

For direct comparator PCMD contrasts, a prompt is eligible only when both methods produce at least two verified responses, and each seed must contain at least 30 such prompts. Some comparator cells therefore have fewer reportable seeds, and no Python comparison meets this threshold at every scale.

RLEP-Dr pool composition. RLEP-Dr replays verified trajectories from a frozen experience pool without balancing solution modes. Among eligible prompts, its pools contain only 1.00–1.02 solution modes on average for Qwen2.5-0.5B and 1.00–1.40 for Falcon3-1B. Two-row replay draws therefore repeat the same solution mode with probability .89–1.00 across the evaluated domains. By contrast, the corresponding Re:Dr buffers at Qwen2.5-0.5B average 1.15–6.88

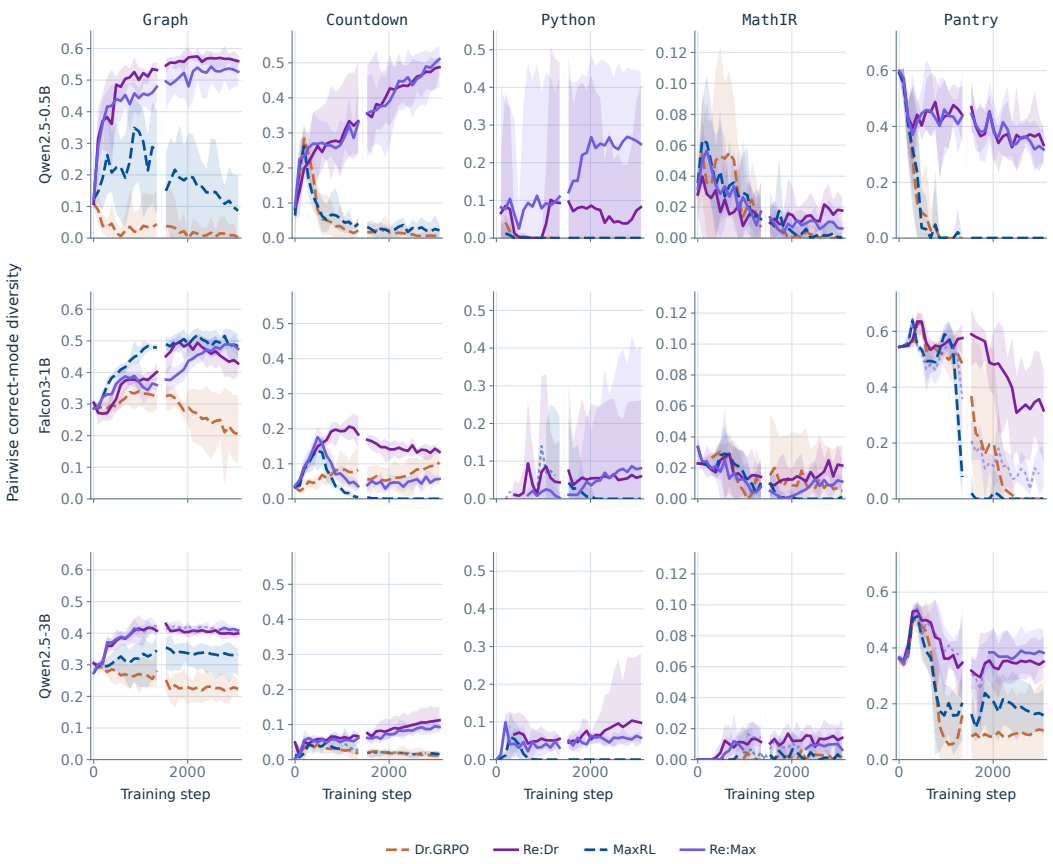

Figure 16: **Replay preserves conditional solution diversity throughout training.** PCMD across eight training passes for Dr.GRPO, Re:Dr, MaxRL, and Re:Max at three model scales. Lines show means over eligible seeds and bands show seed ranges; gaps indicate insufficient verified responses.

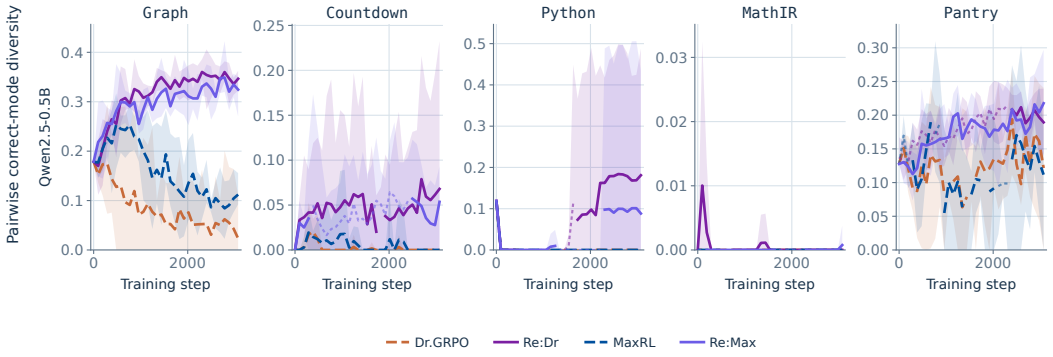

Figure 17: **Replay also preserves diversity on harder Level-2 tasks.** Qwen2.5-0.5B PCMD across training for the four primary methods. Replay produces the clearest separation on Graph, while MathIR remains concentrated under all four methods.

stored solution modes across domains. RLEP-Dr therefore provides a useful replay-of-success baseline whose experience pools contain little multi-mode coverage.

Comparator settings. Comparators use the published or fixed settings used in our evaluation: UCPO uses $\omega = .2$, RLEP-Dr mixes 16 fresh with 2 replay responses, SetPO uses $\lambda = 1$, Semantic-MaxEnt uses $\eta = .10$ and $\kappa = 5$, and our replay methods use $\lambda_{\text{replay}} = .10$ with capacity $M = 16$. GAPO has no free coefficient and its reward is affinely mapped to the control reward range. We use the published or fixed configurations without a benchmark-specific hyperparameter search, so the comparison evaluates the stated methods under a common experimental budget.

Interpreting zero diversity. An undefined `PCMD` comparison indicates insufficient verified responses, whereas .000 denotes measured zero conditional diversity. In `Python`, several methods achieve high `pass@8` while producing only one verified solution mode per eligible prompt, illustrating that correctness can remain high even when conditional solution diversity is zero.

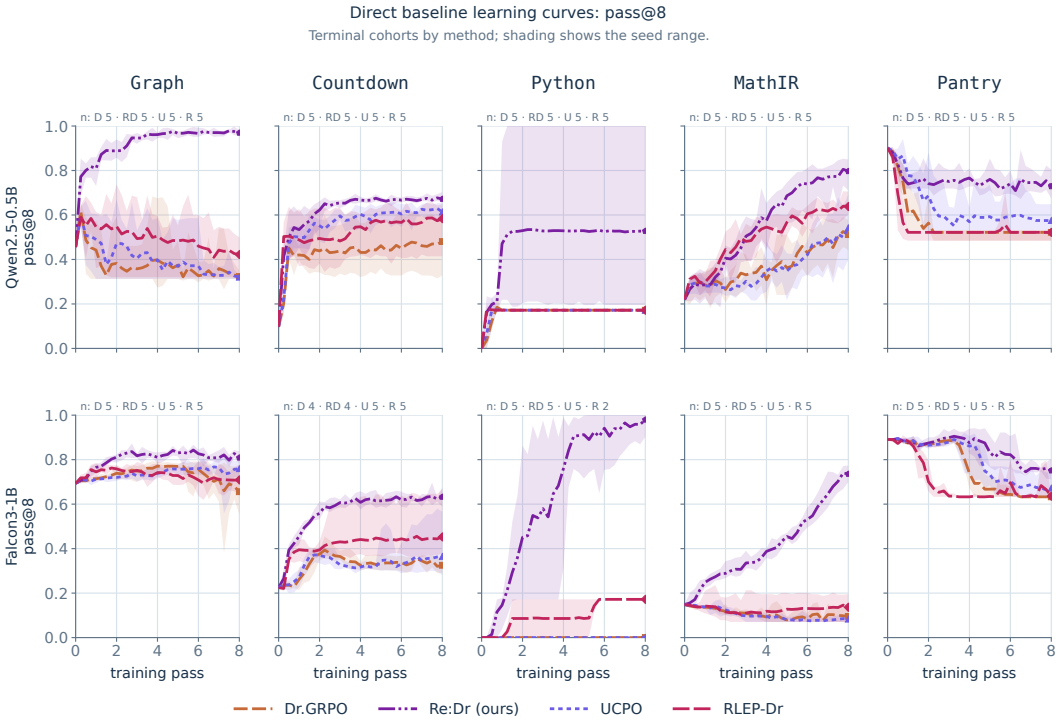

Figure 18: **Re:Dr has the highest terminal mean `pass@8` in every displayed comparison.** Learning curves compare Dr.GRPO, Re:Dr, UCPO, and sparse RLEP-Dr across two model scales and five domains. Lines show seed means and bands show seed ranges.

E.6 Diversity-preserving comparators

We additionally compare against GAPO (Anschel et al., 2025) and SetPO (Li et al., 2026a), which explicitly encourage diversity within the current rollout group. Both are evaluated on Level-1 Qwen2.5-0.5B across five domains and five seeds using a matched Dr.GRPO control. GAPO reweights verified rollouts according to within-group solution-mode frequency, while SetPO rewards each rollout for its marginal contribution to semantic set diversity.

GAPO increases `PCMD` substantially on `Countdown` (+.340) and `PantryPlan` (+.653), while SetPO's clearest diversity gain is on `PantryPlan` (+.266). Against the shared Dr.GRPO control, the five-domain mean `PCMD` effects are +.225 for GAPO, +.066 for SetPO, and +.320 for Re:Max. Re:Max has the largest five-domain mean effect, while GAPO is strongest on `PantryPlan`. SetPO also substantially improves `Python` correctness, raising `pass@8` from .338 to .834.

These methods encourage diversity using responses available in the current rollout group, whereas replay balances verified solution modes accumulated across training. This distinction is especially relevant when the number of valid solution modes greatly exceeds the rollout-group size.

E.7 Reference KL tradeoffs

We compare replay with Dr.GRPO using a reference-policy KL penalty at $\beta \in \{.001, .01, .04, .1, .2, .3\}$ on Qwen2.5-0.5B Level 1. The reference is the initial policy, and all conditions use the same number of optimizer updates and fresh rollouts. KL additionally evaluates the reference policy, while replay scores stored exemplars.

Reference KL can preserve substantial conditional diversity, but its correctness tradeoff depends strongly on the domain and coefficient. In `PantryPlan` at $\beta = .3$, KL reaches `PCMD` $= .931$ and `pass@8` $= .825$, compared with .557 and .797 for Re:Dr. The initial `PantryPlan` policy has `PCMD` $= .91$, consistent with the reference providing a strong diversity target.

The tradeoff is different in `MathIR`. At $\beta = .3$, KL reaches `PCMD` $= .099$ but reduces `pass@8` to .207, compared with .039 and .816 for Re:Dr. In `Graph`, the tested KL coefficients remain below the replay conditions on the reported correctness–diversity tradeoff.

Reference KL anchors the policy to a fixed initial reference, whereas replay rehearses verified solution modes discovered during training. The comparison reveals different tradeoffs: KL can preserve high diversity when the reference itself is broad, while replay achieves a stronger correctness–diversity balance in `Graph` and `MathIR` under the tested settings.

E.8 Uniform versus fresh-frequency replay

This ablation keeps the replay mechanism fixed and changes only how stored solution modes are weighted. Uniform replay assigns equal weight to retained solution modes, whereas frequency-weighted replay assigns weight in proportion to how often each mode appears in verifier-positive fresh rollouts. The buffer construction and update rule are otherwise unchanged, although the resulting policies can accumulate different buffers during training.

We report $B_8 =$ `distinct@8` $-$ `pass@8`, the number of extra verified solution modes per prompt beyond the first success, and $P_8 =$ `pass@8`. Positive contrasts favor uniform weighting. Paired-seed intervals use bootstrap resampling of the matched training seeds.

At Qwen2.5-0.5B, uniform weighting increases extra verified solution modes by $+.318$ per prompt over the five-domain aggregate ($[+.272, +.358]$), with positive within-seed averages in all five seeds. The corresponding `pass@8` effect is $+.010$ ($[-.119, +.139]$), indicating that the diversity gain does not come with a detectable correctness cost. The diversity effect is driven primarily by `Graph`, `Countdown`, and `PantryPlan`.

The same direction appears in several larger-model comparisons. Uniform weighting increases B_8 on Falcon3-1B `Graph` by $+.233$ ($[+.182, +.320]$), on Qwen2.5-3B `Graph` by $+.079$ ($[+.011, +.135]$), and on Qwen2.5-3B `PantryPlan` by $+.207$ ($[+.038, +.377]$). The corresponding correctness effects are not consistently distinguishable from zero.

This comparison isolates the role of balancing replay across stored solution modes. Within the same replay procedure, equal weighting preserves more verified alternatives than weighting modes according to their fresh-rollout frequency.

F Concentration and Replay Mechanisms

F.1 Fresh 64-response evaluation

We reevaluate initial and terminal checkpoints on the 128 held-out Level-1 `Graph` and `PantryPlan` prompts using 64 newly generated responses per prompt. The evaluation covers Qwen2.5-0.5B, Falcon3-1B, and Qwen2.5-3B, with five training seeds for Dr.GRPO, Re:Dr, MaxRL, and Re:Max. For each contrast, `PCMD` is computed on prompts with at least two verified responses in both conditions, and the five paired seed effects are averaged equally.

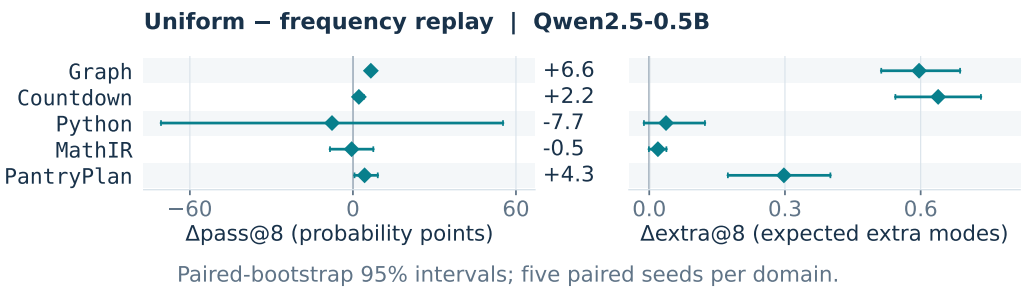

Figure 19: **Uniform weighting preserves more verified alternatives.** Uniform minus frequency-weighted replay at Qwen2.5-0.5B, Level 1. Extra modes are $B_8 = \texttt{distinct@8} - \texttt{pass@8}$. Bars show nominal 95% paired-seed bootstrap intervals.

Dr.GRPO reduces PCMD in all six model–domain comparisons and all 30 seed effects. Replay increases PCMD relative to its matched trained control under both fresh objectives in all twelve model–domain comparisons and all 60 seed effects. Thus the replay diversity advantage persists under a substantially larger fresh evaluation sample and jointly eligible prompt populations.

F.2 Conditional-concentration robustness

As an additional sensitivity analysis, we recompute terminal replay–control differences in conditional concentration using two opposite assignments of disjoint sampling streams. Negative $\Delta\widehat{C}$ means that replay is less concentrated among correct outputs than its matched control. Eligibility is recomputed separately under each stream assignment.

The same qualitative pattern appears under both disjoint stream assignments. At Level 2, seven of ten pooled replay–control comparisons favor replay, one is unchanged, and two PantryPlan comparisons favor the control; several cells have small eligible populations. Together, these disjoint-sampling analyses reinforce the main PCMD result that replay generally reduces concentration among correct outputs.

F.3 Diversity before and after RLVR-only training

We compare Dr.GRPO and GRPO before training and after eight passes across three model scales and five domains. We decompose distinct@8 into pass@8 and the number of additional verified modes beyond the first success.

Across both objectives, the average number of additional verified modes decreases at every tested model scale. The decline is approximately 98–99% at Qwen2.5-0.5B and 42–72% at the larger scales. This loss of breadth can occur even when success improves: pass@8 does not consistently decrease, and aggregate distinct@8 can increase when gains in correctness outweigh losses in additional modes.

Where conditional diversity is measurable at both endpoints, PCMD shows the same general pattern. Its domain-average decline is approximately 99% at Qwen2.5-0.5B and 51–65% at the larger scales, although individual domain–model combinations need not decrease. Thus the reduction in observed solution breadth is not explained solely by changes in the probability of producing a correct response.

Initial evaluations are sampled separately for each training method, so small differences between their pass-0 estimates can reflect sampling variability. These evaluations measure ob-

Table 6: **Replay generally lowers conditional concentration under disjoint sampling splits.** Counts summarize Level-1 replay–control comparisons across three model scales, five domains, and two objectives. One Falcon3-1B Python comparison is undefined because too few verified responses are available.

Estimate	Replay lower	No change	Replay higher
Pooled estimate	27/29	1/29	1/29
Disjoint split 0	25/29	3/29	1/29
Disjoint split 1	24/29	3/29	2/29

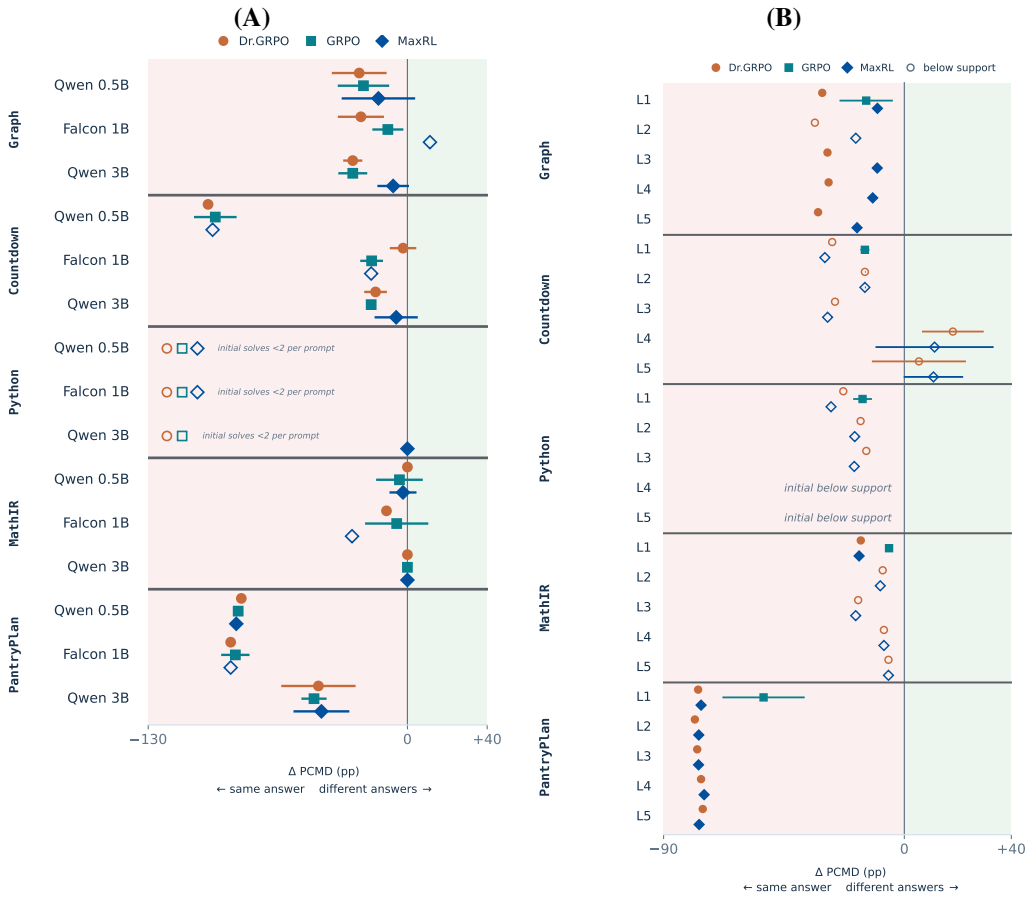

Figure 20: **`Graph` and `PantryPlan` lose conditional solution diversity across scales and evaluation levels.** **(A)** Initial-to-final changes in `PCMD` at Level 1 across three model scales and no-replay objectives. **(B)** Level-1-trained policies evaluated on Levels 1–5, with higher levels measuring transfer. Filled marks show well-supported comparisons, open marks indicate limited support, and bars show nominal 95% intervals where available.

Table 7: **Fresh reward information is sparse in several control runs.** Entries are the fraction of Dr.GRPO rollout groups containing both correct and incorrect responses, averaged over five seeds.

Domain	Mixed groups
`Graph`	.144
`Countdown`	.140
`Python`	.015
`MathIR`	.364
`PantryPlan`	.123

served solution breadth; claims throughout concern sampled concentration rather than literal zero probability for unobserved modes.

F.4 Sparse task gradients and output repetition

For Dr.GRPO, a rollout group with identical rewards has zero fresh task-gradient contribution because all within-group advantages vanish. We therefore measure the fraction of training groups containing both correct and incorrect responses across 3,072 Qwen2.5-0.5B updates.

The effect is most extreme in `Python`, where only 1.5% of control groups are mixed and therefore 98.5% have zero fresh task advantage. Replay can still provide a likelihood gradient on these updates and can alter subsequent sampling so that future groups become informative.

The `Python` controls also exhibit strong output repetition. At initialization, eight-response groups contain an average of 7.99 distinct completion strings per prompt, but all five control seeds later

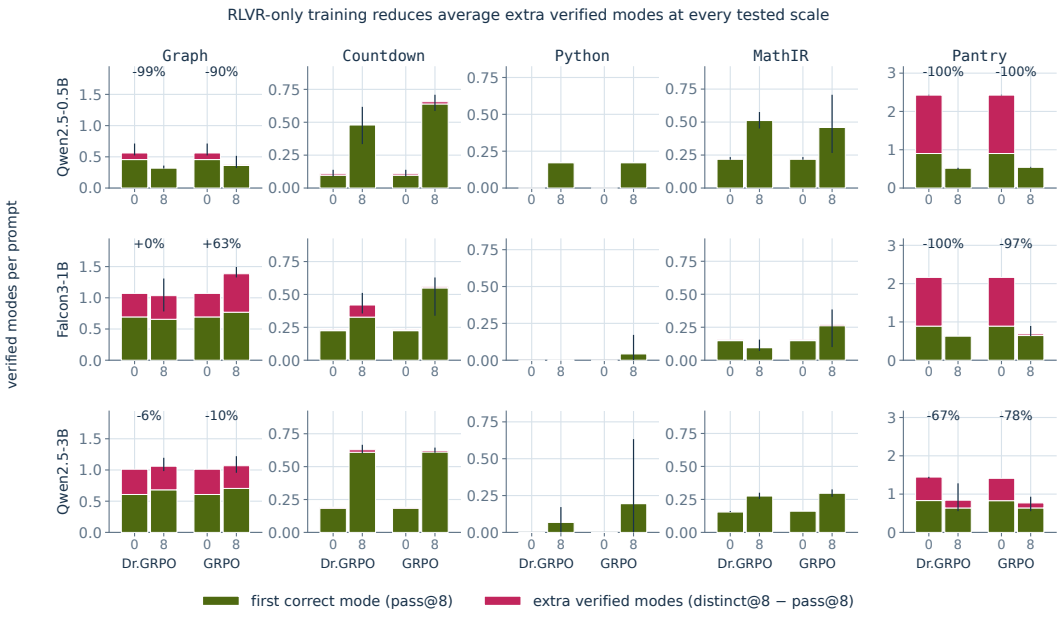

Figure 21: **RLVR-only training reduces additional verified modes across scales.** Bars decompose `distinct@8` into `pass@8` and additional verified modes before training and after eight passes for Dr.GRPO and GRPO across three model scales and five domains. Whiskers show ranges across five training seeds.

Table 8: **Replay gains persist when controls are learning.** Mean Re:Dr minus Dr.GRPO effects across increasingly restrictive subsets of the 15 model–domain comparisons.

Control subset	n	Δ`pass@8`	Δ`PCMD`
All comparisons	15	+.352	+.189
Increasing control `pass@8`	9	+.379	+.118
Nonzero initial `pass@8`, $\geq 10\%$ mixed groups	6	+.247	+.136

produce a single repeated completion for every prompt and remain at that level through the terminal evaluation. At the end of training, eight-response groups contain 1.00 distinct completion per prompt for Dr.GRPO and 1.97 for Re:Dr. The sampled-output collapse is therefore pronounced under the tested Dr.GRPO setting. The decoding and group-size controls below test whether straightforward inference-time or sampling-budget changes recover the lost breadth.

F.5 Replay gains with active controls

Control and replay runs share the learning-rate schedule, initialization, optimizer, prompt stream, update count, and fresh-rollout budget. Per-response correctness increases from the initial to terminal checkpoint in all 15 control comparisons, confirming that the controls learn under these schedules. Fresh reward information is nevertheless sparse in some settings, most notably Falcon3-1B `Python`, where only .9% of rollout groups contain both correct and incorrect responses.

Table 8 repeats the replay comparison on increasingly restrictive subsets of active controls. Replay remains beneficial among the nine comparisons in which control `pass@8` increases and among the six comparisons with nonzero initial `pass@8` and at least 10% mixed rollout groups. All six comparisons in the latter subset have positive replay effects on both `pass@8` and `PCMD`. Thus the replay advantage persists when the comparison is restricted to controls with clear evidence of task learning and nontrivial fresh reward signal.

G Human-Authored Programming Evaluation

We additionally test verified replay on human-authored constructive programming problems whose correctness checkers were not designed for this study. We use 21 Codeforces problems from

CodeContests+ and CodeContests-O with released output checkers and frozen hidden-input suites. Eight problems are used for training and thirteen for development. Correctness is determined by the released checker, while solution-mode identity is defined by the canonical witness signature produced across the frozen suite. Because the suites are finite, these modes characterize behavior on the frozen checker suite rather than semantic program equivalence.

We train Qwen2.5-Coder-7B-Instruct in a 2×2 factorial crossing Dr.GRPO versus MaxRL with replay off versus on, using five paired seeds per arm and 128 updates. Replay uses the same mechanism as in the main experiments: one verified exemplar per discovered solution mode, uniform rehearsal, and a capacity of 16 modes per prompt. The controls execute the same buffer construction, scheduling, and exemplar scoring with the replay gradient set to zero.

Replay produces a small but consistent training-side increase in the fraction of fresh rollouts that verify. Across matched seeds, the verified fraction of fresh rollouts is $+.014$ higher for Re:Dr than for Dr.GRPO, with a 95% interval of $[+.005, +.023]$, and $+.014$ higher for Re:Max than for MaxRL, with interval $[-.002, +.031]$. Pooling the ten matched replay–control seed pairs gives the same $+.014$ effect, with interval $[+.007, +.021]$ and a positive effect in nine of ten pairs.

The replay mechanism transfers unchanged to these human-authored constructive programming tasks and produces a small, consistent training-side correctness improvement. This extends the method beyond the synthetic MODEBENCH task generators while retaining executable verification and solution-mode identity.

Part IV. What diversity represents

H Prompt-Guidance Ablation

We compare benchmark and neutral prompts on the same 32 `Python`, `MathIR`, and `PantryPlan` problems at Levels 2–3, with eight responses per condition. The neutral prompts remove task-specific solution guidance while leaving the problem, answer format, verifier, and remaining interface unchanged. Because the `Python` intervention also removes an executable example and associated divisor guidance, it should be interpreted as a combined prompt-interface change rather than an isolated hint removal.

Table 9: **Removing guidance leaves `MathIR` concentrated but sharply reduces `Python` correctness.** Results use matched Qwen2.5-0.5B checkpoints and strict verification on 32 paired problems per level. `PCMD` is reported only where sufficient jointly eligible problems remain.

Model	Cell	`pass@8`: benchmark → neutral	`PCMD`: benchmark → neutral
Dr.GRPO	`Python` L2	.812 → .000	–
Dr.GRPO	`Python` L3	1.000 → .000	–
Re:Dr	`Python` L2	.887 → .556	.231 → .173
Re:Dr	`Python` L3	1.000 → .494	.256 → .323
Dr.GRPO	`MathIR` L2	.531 → .506	.000 → .000
Re:Dr	`MathIR` L2	.994 → .975	.000 → .000
Re:Dr	`MathIR` L3	.312 → .275	.000 → .000

Removing the operation-order guidance does not recover alternative verified `MathIR` solution modes. Every eligible trained `MathIR` comparison has `PCMD` $= .000$ under both benchmark and neutral wording at Levels 2–3. At Level 2, Re:Dr remains highly successful under neutral wording, with `pass@8` .975 compared with .994 under benchmark wording.

The `Python` intervention instead causes a large loss of correctness, particularly for Dr.GRPO. Re:Dr retains neutral-wording `pass@8` of .556 and .494 at Levels 2 and 3, but its two reportable `PCMD` changes have opposite signs. The `Python` result therefore shows that prompt interface can materially change both correctness and the observed diversity available for comparison.

The two-seed `PantryPlan` comparison shows smaller, mixed effects and is reported as a supplementary prompt-sensitivity check.

Hosted prompt-guidance sensitivity. We also evaluate benchmark and neutral wording with 64 responses on 16 `MathIR` problems per level for fixed hosted deployments. Among problems with at least eight verified responses under both wordings, neutral wording increases eight-correct-draw solution breadth for GPT-5.4 by .794 at Level 2 and .204 at Level 3, and for GPT-5.6 Sol by .872 and .860, respectively. The corresponding Grok 4.3 comparisons are inconclusive. Thus some hosted-model concentration is sensitive to prompt guidance, even though the local trained `MathIR` checkpoints remain concentrated after the benchmark hint is removed.

I Inference-Time Decoding Controls

We test whether the low solution diversity of RLVR-only policies can be recovered by changing inference-time decoding at fixed checkpoints.

Temperature. We evaluate eight-pass Qwen2.5-0.5B Dr.GRPO checkpoints at temperatures $T \in \{.5, .7, 1.0, 1.3, 1.6, 2.0\}$ with top-$p = 1$ and the same prompts and response budget. Across the six temperatures, the largest control `PCMD` is .021 in `Graph`, .016 in `Countdown`, .002 in `MathIR`, and .004 in `PantryPlan`. For comparison, Re:Dr at $T = 1$ reaches .562, .485, .020, and .330, respectively. `Python` controls do not produce enough verified responses to estimate conditional diversity under our reporting threshold. Thus temperature changes alone do not recover the solution diversity retained by replay in the domains where the comparison is defined.

Sampling budget and nucleus sampling. We repeat the control analysis on a separate twelve-pass cohort across eleven decoding settings spanning temperature, top-p, and response-group size.

Increasing the group size from $K = 8$ to $K = 32$ changes control PCMD by at most .010 across matched settings, and no $K = 32$ control mean exceeds .054. The broader grid likewise leaves control diversity low across the eligible domains. These comparisons change inference-time sampling at fixed checkpoints and therefore do not test changes to rollout sampling during training.

Overall, the tested decoding interventions do not eliminate the diversity gap between RLVR-only and replay-trained policies. Missing PCMD estimates indicate insufficient verified responses rather than evidence that no alternative correct modes exist.

J Sampling-Budget Ablation

J.1 Sampling-budget design

We test whether small sampling budgets understate observed solution breadth by collecting up to 64 responses per prompt in the local evaluation and 512 responses per prompt in a separate hosted evaluation. For a response pool of size N with c verified responses and solution-mode counts n_m, the expected pass@k and distinct@k of a uniformly sampled k-response subset are

$$P_k = 1 - \frac{\binom{N-c}{k}}{\binom{N}{k}}, \qquad D_k = \sum_m \left[1 - \frac{\binom{N-n_m}{k}}{\binom{N}{k}}\right].$$

Incorrect, empty, refused, and truncated responses remain in the sampling denominator.

To compare breadth at a fixed number of verified responses, we also use

$$R_m = \sum_j \left[1 - \frac{\binom{c-n_j}{m}}{\binom{c}{m}}\right],$$

computed only when at least m verified responses are available. These quantities describe discovery within finite response pools; throughout this section, breadth refers to modes observed under the stated sampling budget.

J.2 Discovery at larger sampling budgets

We extend the local-model evaluation from eight to 64 responses per prompt on 16 held-out problems per domain and level. The trained checkpoints were optimized on Level 2, so Level 3 evaluation measures transfer.

MathIR. Increasing the budget from eight to 64 responses reveals essentially no additional solution modes after training. At Level 2, every measured trained condition produces at most one verified solution mode per checkpoint and problem. At Level 3, only one Dr.GRPO checkpoint–problem pair produces a second mode. The near-single-mode behavior observed at eight draws therefore largely persists at 64.

Python. For Re:Dr under benchmark wording, increasing the budget from eight to 64 responses adds .058 observed solution modes per prompt at Level 2 and .153 at Level 3. Neutral wording reveals more additional solutions but also changes correctness substantially, so the magnitude of discovery depends on both sampling budget and prompt interface.

PantryPlan. Across trained checkpoints, increasing the budget from eight to 64 responses adds approximately .42–1.02 observed solution modes per prompt. The ordering between Dr.GRPO and Re:Dr can change with sampling budget in this two-seed comparison, so an eight-response result need not determine which checkpoint exposes more modes at substantially larger budgets.

Overall, larger budgets reveal additional modes in some domains while leaving the strongest concentration patterns intact. The result is therefore not an artifact of using only eight evaluation draws.

J.3 Discovery at 512 draws

We separately evaluate GPT-5.6 Sol on 32 problems from each of the five domains at Levels 1–3, with 512 responses per prompt. This yields 480 prompt pools and 245,760 responses.

Even at this budget, observed breadth remains far below the certified or enumerated solution support in several domains. Mean `distinct@512` is 2.53, 2.63, and 1.72 for `Graph` across Levels 1–3. For `MathIR`, it is 1.91, 1.00, and .97, despite at least five certified solution modes per problem. For `Python`, it is 3.75, 1.44, and 1.34, compared with enumerated averages of roughly 222, 336, and 355 solution modes. For `PantryPlan`, it ranges from 4.88 to 6.91, compared with certified-support means near 16–17.

Late discovery also differs by domain. Doubling the budget from 256 to 512 responses adds approximately .14–.27 modes in `Graph`, at most .07 in Level-2/3 `Python`, approximately zero in Level-2/3 `MathIR`, and .6–.7 in `PantryPlan`. Thus larger budgets continue to reveal alternatives in some domains while leaving others close to single-mode behavior, even after 512 draws per prompt.

K Inference-Time Alternatives and Mode Definitions

K.1 Complementarity across hosted models

Seven hosted deployments each produce eight responses to the same 1,920 prompts. For each of the 21 model pairs, we compare a mixed portfolio containing four responses from each model with the two constituent eight-response portfolios.

Every pair increases expected distinct solution modes relative to the average of its two constituents, with gains ranging from .115 to .266 modes. Ten pairs exceed both constituent point estimates, and nine have intervals above both. When the comparison instead fixes the portfolio to four verified responses, with two drawn from each model, every pair again exceeds the average constituent by .073–.177 modes.

The gains persist under coarser solution-mode definitions and formatting-normalized grading. Thus different deployments expose reproducibly complementary sets of verified solutions under the tested sampling budgets.

K.2 Sensitivity to solution-mode identity

We repeat selected analyses under coarser task-relevant definitions of solution identity. For `Graph`, we group solutions related by global color renaming while respecting anchored colors. For `Python`, we identify opposite members d and n/d of the same unordered factor pair. For `Countdown`, we ignore associative reorderings of addition and multiplication while preserving operands and subtraction or division boundaries. The `MathIR` and `PantryPlan` definitions are unchanged.

Coarsening reduces observed solution counts and increases concentration, as expected. The largest sensitivity occurs in `Python`: under benchmark wording, the Re:Dr minus Dr.GRPO `distinct@8` gain falls from .431 to .075 at Level 2 and from .419 to .000 at Level 3. Under neutral wording, the corresponding coarse-mode gains remain .556 and .500. The measured diversity gain therefore depends in part on what distinctions are considered meaningful, particularly for `Python`.

L Portfolio Survival under Withdrawn Options

We test whether retaining multiple verified solution modes makes a sampled portfolio more useful after the task changes. For each prompt, we withdraw one available option: a color choice in `Graph`, an operation in `Countdown`, a divisor value in `Python`, an action in `MathIR`, or an ingredient in `PantryPlan`. We restrict the primary analysis to binding withdrawals that remove at least one certified solution mode while leaving at least one certified mode feasible.

To reduce dependence on overall correctness, we compare survival after sampling a fixed number of verified responses. At four verified draws, replay improves survival most strongly in `Graph`, `Countdown`, `Python`, and `PantryPlan`, while `MathIR` changes little.

L.1 Two withdrawals at once

We additionally remove pairs of binding options that leave at least one certified solution mode feasible. The pattern remains strongest in `Graph`, `Python`, and `PantryPlan`, while the Re:Max

Table 10: **Replay generally improves survival after one or two withdrawn options.** Entries are replay minus matched control survival in percentage points at four verified draws.

	One withdrawal		Two withdrawals	
Domain	Re:Dr	Re:Max	Re:Dr	Re:Max
`Graph`	19.65	10.99	23.65	12.51
`Countdown`	9.58	0.10	6.11	-4.87
`Python`	7.67	5.68	10.10	8.16
`MathIR`	0.14	0.09	0.20	0.12
`PantryPlan`	9.78	6.61	7.23	4.94

`Countdown` point estimate reverses. These results show that survival gains can persist under a more severe change but are not uniform across domains and objectives.

L.2 Decoding settings as a substitute

We compare replay with eleven inference-time control settings spanning temperature, nucleus sampling, and response-group size on a separate Level-1 cohort. For each domain, we select the control setting with the highest four-verified-draw survival. Replay exceeds this selected control in all five domains, with paired gains ranging from 1.37 to 36.97 percentage points and pointwise intervals above zero. Thus the tested inference-time decoding changes do not reproduce the portfolio-survival gains of replay in this cohort.

L.3 Recovery after a withdrawal

We also allow up to eight additional responses when the initial eight-response portfolio contains no surviving solution. This evaluation uses Level-1 Qwen2.5-0.5B Dr.GRPO and Re:Dr checkpoints from two matched seeds and a common recovery procedure across all five domains.

Table 11: **Replay improves ordinary-sampling recovery after a withdrawal.** Entries are Re:Dr minus Dr.GRPO on the same withdrawn tasks. Saved and resolved-by-eight differences are percentage points; calls are additional recovery responses.

Domain	Δ saved	Δ by 8	Δ calls
`Graph`	67.58	68.36	-5.37
`Countdown`	6.76	8.11	-0.55
`Python`	55.86	63.67	-4.67
`MathIR`	20.54	20.98	-1.68
`PantryPlan`	16.41	20.70	-1.44

All ordinary-sampling recovery intervals exclude zero in the favorable direction. Temperature-tuned portfolios show the same qualitative pattern. Diversity prompting is less consistent and reverses the comparison in `PantryPlan`, so the recovery advantage depends on how the initial portfolio is constructed.

These withdrawal experiments connect sampled solution breadth to a concrete downstream benefit: retaining or recovering a feasible response after the task changes. Because replay changes the trained policy as a whole, the survival gain should be interpreted as a property of replay-trained portfolios rather than as a mediation estimate for diversity alone.

Part V. Hosted deployment analyses

M Inference-Only Concentration in GPT-5.6 Sol

We evaluate GPT-5.6 Sol on 128 held-out prompts from each of five domains at Levels 1–3, with eight stateless responses per prompt. The resulting 15,360 responses use medium reasoning, an 8,192-token output limit, no tools, and no conversation state. These inference-only measurements characterize one hosted deployment and are separate from the matched training interventions.

Table 12: **GPT-5.6 Sol produces few verified solution modes at each benchmark level.** Values average the five domains equally under strict grading. P is `pass@8`, D is mean `distinct@8`, and C is correct-pair collision.

Level	P (%)	D	C (%)
1	95.8	1.71	74.0 [71.9, 76.1]
2	77.5	1.16	82.3 [80.3, 84.4]
3	80.3	1.18	84.5 [82.8, 86.2]

Concentration can remain high even when correctness is nearly perfect. On Level-3 `Graph`, all 1,024 responses verify, yet mean `distinct@8` is 1.16 and 95.1% of correct-response pairs share the same solution mode, compared with a uniform reference near 19%. Level-2 and Level-3 `MathIR` are similarly concentrated, while `PantryPlan` exhibits greater conditional diversity.

Strict formatting affects observed success most strongly in `Python`, so the hosted results characterize the tested deployment and interface. The concentration pattern nevertheless appears across multiple domains and benchmark levels.

N Comparison across Hosted Deployments

We compare seven hosted deployments on the same five executable domains, three levels, and eight responses per prompt. The common-prompt cohort contains 107,520 responses. Provider interfaces, reasoning implementations, sampling defaults, and compute differ across systems, so the comparison characterizes deployment-level solution concentration rather than isolating model identity.

N.1 Hosted-model concentration

Table 13 reports promptwise `PCMD` under strict grading and benchmark wording. Each cell averages prompts with at least two verified responses, and cells with fewer than thirty eligible prompts are omitted.

GPT-5.6 Sol reaches `PCMD` = .000 on `MathIR` at Levels 2–3 and .049 on Level-3 `Graph`, while `PantryPlan` reaches .531, .386, and .455 across Levels 1–3. Across the thirteen domain–level cells reportable for all seven deployments, macro `PCMD` ranges from .142 to .320. The corresponding effective-mode transforms range from 1.17 to 1.47. Removing `MathIR` raises the summaries but leaves the same qualitative concentration pattern, with the largest transformed value reaching 1.58. Because these summaries depend on the averaging population, we use them to characterize concentration rather than to rank deployments.

A separate common-prompt comparison excludes `Python`, whose interface differs for one deployment. Under formatting-normalized grading, all seven deployments solve most prompts while producing relatively few verified solution modes.

Cross-level summaries combine changes in task population and guidance, so we use them to characterize concentration across benchmark regimes rather than as an isolated difficulty intervention.

N.2 Python prompt-interface sensitivity

We test the same `Python` problems under the benchmark interface and a direct-expression interface without a system message. The effect of this interface change differs sharply across the two deployments on which it is tested.

Table 13: **Hosted deployments remain concentrated among verified solutions.** Macro `PCMD` averages the thirteen domain–level cells reportable for every deployment. Cells require at least thirty prompts with two verified responses.

			Macro `PCMD` by level		
Deployment	Macro `PCMD`	$N_{\text{eff}}^{(2)}$	L1	L2	L3
DeepSeek V4 Pro	.320	1.47	.289	.319	.232
Kimi K3	.302	1.43	.374	.307	.253
GPT-5.4	.264	1.36	.338	.247	.224
Grok 4.3	.239	1.31	.275	.179	.171
GPT-5.6 Sol	.226	1.29	.256	.229	.183
Claude Opus 4.8	.215	1.27	.217	.229	.168
Claude Opus 5	.142	1.17	.140	.164	.121

Table 14: **Hosted deployments produce few verified modes on common prompts.** Equal-domain means over `Graph`, `Countdown`, `MathIR`, and `PantryPlan`, with 128 prompts per domain and level and eight responses per prompt. `pass@8` is prompt success and # modes is mean `distinct@8`, including unsolved prompts.

	Level 1		Level 2		Level 3	
Deployment	`pass@8`	# modes	`pass@8`	# modes	`pass@8`	# modes
GPT-5.6 Sol	97.5	1.90	98.2	1.59	99.4	1.54
Claude Opus 5	97.9	1.42	99.6	1.45	99.8	1.35
GPT-5.4	97.7	2.00	98.0	1.68	99.4	1.62
Grok 4.3	99.4	2.12	100.0	1.70	100.0	1.68
Kimi K3	98.0	2.26	100.0	1.95	100.0	1.79
Claude Opus 4.8	98.4	1.70	100.0	1.75	99.8	1.56
DeepSeek V4 Pro	98.0	2.16	98.8	2.04	99.6	1.75

Table 15: **`Python` prompt-interface effects differ across deployments.** Entries show benchmark $\rightarrow$ direct-expression results under formatting-normalized grading. Accuracy is per-response verified accuracy, and D is mean `distinct@8`.

Deployment	Level	Refusals	Accuracy (%)	D
Claude Opus 5	1	$409 \rightarrow 0$	$60.0 \rightarrow 100.0$	$1.12 \rightarrow 1.49$
Claude Opus 5	2	$1021 \rightarrow 0$	$.3 \rightarrow 100.0$	$.02 \rightarrow 1.38$
Claude Opus 5	3	$1024 \rightarrow 3$	$.0 \rightarrow 99.7$	$.00 \rightarrow 1.41$
GPT-5.6 Sol	1	$0 \rightarrow 0$	$89.8 \rightarrow 20.9$	$1.36 \rightarrow .41$
GPT-5.6 Sol	2	$0 \rightarrow 0$	$58.7 \rightarrow 16.7$	$1.15 \rightarrow .45$
GPT-5.6 Sol	3	$0 \rightarrow 0$	$56.3 \rightarrow 19.3$	$1.13 \rightarrow .48$

For Claude Opus 5, the direct-expression interface largely removes provider refusals and sharply increases verified success. For GPT-5.6 Sol, the same interface change lowers both verified accuracy and sampled solution breadth. A non-verifying LaTeX wrapper also appears more frequently in the GPT-5.6 Sol direct-expression condition, so part of that gap is associated with output formatting. The GPT-5.6 Sol interface comparison uses a separate deployment configuration, so inference focuses on the within-deployment benchmark-to-direct change.

Formatting normalization corrects typography before applying the same executable verifier. It does not alter mathematical values, program logic, or the definition of a solution mode.

N.3 Reasoning-control comparison

Five deployments answer the same 480 prompts under medium reasoning and with explicit reasoning disabled through the provider interface. Each condition uses 32 prompts in each domain–level cell and eight responses per prompt.

All five deployments have lower overall `pass@8` and `distinct@8` when explicit reasoning is disabled, but the change in conditional diversity is not uniform. The reasoning-control intervention changes the provider-exposed reasoning setting rather than internal compute directly. A separate DeepSeek V4 Pro comparison gives strict-grading `PCMD` of .282 with medium reasoning

Table 16: **Disabling explicit reasoning lowers sampled coverage for all five tested deployments.** Entries show medium reasoning → disabled reasoning. `PCMD` uses jointly eligible prompts and therefore need not move with raw mode counts.

Deployment	`pass@8` (%)	# modes	`PCMD`
GPT-5.6 Sol	98.1 → 75.2	1.62 → .99	.226 → .145
GPT-5.4	98.5 → 74.6	1.85 → 1.18	.255 → .224
Grok 4.3	100.0 → 86.5	1.70 → 1.50	.154 → .364
Kimi K3	99.0 → 93.5	2.06 → 1.62	.293 → .426
Claude Opus 4.8	99.4 → 97.1	1.62 → 1.53	.202 → .209

and .369 with reasoning disabled on 437 jointly eligible prompts. Claude Opus 5 is omitted because the provider's nominally disabled condition still returns thinking tokens.

N.4 Temperature sensitivity across deployments

We first sweep requested temperature for GPT-5.6 Sol with explicit reasoning disabled on the same 480 prompts. Formatting-normalized `pass@8` and sampled solution breadth both increase over much of the sweep, while per-response accuracy decreases.

Table 17: **GPT-5.6 Sol exposes more sampled modes at higher requested temperature in this sweep.** Each temperature uses the same 480 prompts and eight responses per prompt. A is per-response verified accuracy.

T	`pass@8` (%)	`distinct@8`	A (%)
0.0	65.6	.673	62.1
0.5	72.3	.840	61.2
1.0	75.2	.963	59.1
1.5	79.8	1.125	57.7
2.0	79.4	1.146	55.9

Between $T = 0$ and $T = 2$, `pass@8` increases by 13.8 percentage points ($[10.8, 16.7]$) and `distinct@8` by .473 modes ($[.417, .529]$). This increase in finite-sample coverage accompanies a decrease in per-response accuracy from 62.1% to 55.9%.

We also compare requested temperatures $T = 1.0$ and $T = 1.5$ for Grok 4.3 and Kimi K3 on 120 common prompts.

Table 18: **Temperature effects are deployment-dependent.** Entries show $T = 1.0 \rightarrow 1.5$ under strict grading.

Deployment	Accuracy (%)	`distinct@8`
Grok 4.3	98.65 → 96.35	1.608 → 1.658
Kimi K3	96.56 → 21.25	1.933 → .983

For Grok 4.3, the mode-count change is $+.050$ with interval $[-.058, +.158]$, while accuracy decreases by 2.29 percentage points. For Kimi K3, 696 of 960 responses terminate at the token limit at $T = 1.5$, compared with none at $T = 1.0$, accompanying the large drops in both accuracy and sampled modes. Temperature can therefore increase sampled solution coverage in some settings, but it is not a deployment-independent substitute for preserving solution modes during training.

N.5 Native provider outcomes

Provider metadata distinguish declared refusals from executable verification failures. In the hosted cohort, Claude Opus 5 is the only deployment with provider-declared refusals, totaling 2,522 of 15,360 responses. Most occur in `Python`: 409 of 1,024 responses at Level 1, 1,021 at Level 2, and all 1,024 at Level 3; the remaining 68 occur in `MathIR`. These refusals explain why several Claude Opus 5 conditional-diversity cells are undefined. Other deployments have no provider-declared refusals in this cohort, although empty responses, truncations, and other verification failures still count as unsuccessful outputs.

Part VI. Mechanism and efficiency

O Mode Collapse and Verified Replay

The winner-take-all limit for binary-reward policy gradient among equally rewarded outcomes is established by Sinha et al. (2026) and Lochab et al. (2026). We use a simple solution-mode model to connect this result to the verified solution modes studied in this paper and analyze how replay changes the dynamics.

O.1 Solution-mode model and the known winner-take-all limit

We first show that, under the idealized solution-mode model, Dr.GRPO and centered MaxRL induce the same mean update direction and therefore inherit the known winner-take-all limit. For one prompt, let $\mathcal{C}$ denote the set of correct solution modes and $\mathcal{N}$ the incorrect outcomes. Each mode $a \in \mathcal{C} \cup \mathcal{N}$ has an independent logit z_a and probability $p_a = \mathrm{softmax}(z)_a$. Let

$$P = \sum_{c \in \mathcal{C}} p_c, \qquad q_c = \frac{p_c}{P},$$

so that P is the probability of a correct response and q describes how that probability is distributed among correct modes.

Assumption O.1 (Solution-mode mean-flow model)**.** We study infinitesimal expected-gradient updates in the independent logits z, with common response-length scaling, no reference-KL or entropy regularizer, finite initial logits, $0 < P(0) < 1$, and rollout group size $G \geq 2$.

This solution-mode model isolates the mean binary-reward dynamics. Neural policies additionally couple solution modes through shared parameters and introduce response-length effects, finite PPO steps, AdamW preconditioning, momentum, and clipping.

Binary-reward objectives share the same mean direction. For a group of G on-policy samples, let $R_i \in \{0, 1\}$ denote correctness and $R = \sum_i R_i$.

Consider any bounded, detached advantage $A(R_i, R)$ that depends only on the group's binary rewards. The corresponding on-policy score estimator is

$$\widehat{g}_A = \frac{1}{G} \sum_{i=1}^{G} A(R_i, R) \nabla_z \log p_{Z_i}.$$

Lemma O.2 (Binary-reward mean update)**.** *At the on-policy point,*

$$\mathbb{E}[\widehat{g}_A] = c_A(P) \nabla_z P, \tag{9}$$

where $c_A(P)$ is continuous and bounded on $[0, 1]$. For Dr.GRPO,

$$c_{\mathrm{Dr}}(P) = \frac{G - 1}{G},$$

while for the centered MaxRL estimator,

$$c_{\mathrm{MaxRL}}(P) = \frac{1 - (1 - P)^{G-1}}{P},$$

with continuous extension $c_{\mathrm{MaxRL}}(0) = G - 1$. Both coefficients are positive on $(0, 1)$.

Proof. Conditioning on whether a response is correct gives

$$\mathbb{E}[\nabla_z \log p_Z \mid R = 1] = \nabla_z \log P, \qquad \mathbb{E}[\nabla_z \log p_Z \mid R = 0] = \nabla_z \log(1 - P).$$

Because the advantage depends only on the binary reward vector, taking the expectation therefore produces a scalar multiple of $\nabla_z P$. For fixed G and bounded advantage values, this scalar is a finite sum of binomial probabilities and is therefore continuous and bounded on $[0, 1]$. The stated Dr.GRPO and MaxRL coefficients follow by substituting their respective advantages. □

Thus, in this solution-mode model, Dr.GRPO and centered MaxRL change the speed of the mean dynamics but not their direction.

Known winner-take-all limit. For independent Euclidean logits, dynamics of the form $\dot{z} = c(P)\nabla_z P$ with $c(P) > 0$ reduce, after a positive change of time, to the replicator dynamics studied by Sinha et al. (2026) and Lochab et al. (2026). Consequently, if one correct mode $c^\star$ initially has strictly greater probability than every other correct mode, then

$$P(t) \longrightarrow 1, \qquad q_{c^\star}(t) \longrightarrow 1.$$

We use this known result as the baseline solution-mode collapse mechanism for the replay analysis below; we do not reprove it. This asymptotic result provides the idealized collapse mechanism analyzed below.

O.2 Rare modes receive vanishing on-policy signal

Once a solution mode becomes rare, the policy rarely samples it, so fresh on-policy RLVR receives little direct signal about that mode.

Lemma O.3 (Vanishing signal for rare modes)**.** *Under Assumption O.1, let mode b have probability p_b, and draw G responses independently from the current policy. Its probability of appearing at least once is*

$$1 - (1 - p_b)^G \le G p_b,$$

which approaches zero as $p_b \to 0$.

Moreover, let the detached advantages satisfy $|A_i| \le B$ for some fixed B. At the on-policy point, the expected absolute logit-gradient contribution for mode b satisfies

$$\mathbb{E}|\widehat{g}_b| \le 2Bp_b(1 - p_b),$$

and therefore also approaches zero as $p_b \to 0$.

Proof. The sampling bound follows from a union bound over the G draws. For

$$\widehat{g}_b = \frac{1}{G}\sum_{i=1}^{G} A_i\big(\mathbf{1}\{Z_i = b\} - p_b\big),$$

the triangle inequality and $|A_i| \le B$ give

$$\mathbb{E}|\widehat{g}_b| \le B\,\mathbb{E}|\mathbf{1}\{Z = b\} - p_b| = 2Bp_b(1 - p_b).$$

□

Replay removes this dependence on rediscovery. For a fixed replay target with positive weight w_b for a mode previously discovered and retained in the replay buffer, the replay contribution to its solution-mode logit is $\rho(w_b - p_b)$, which approaches $\rho w_b > 0$ as $p_b \to 0$. Thus, unlike the fresh on-policy signal, replay can continue to reinforce a stored mode after the current policy almost never samples it.

O.3 Fixed-buffer replay preserves stored modes

Let $\mathcal{B} \subseteq \mathcal{C}$ be a nonempty fixed buffer of k verified solution modes. Uniform replay assigns one stored exemplar to each mode and, in the solution-mode model, minimizes

$$R_{\mathcal{B}}(z) = -\frac{1}{k}\sum_{b\in\mathcal{B}} \log p_b. \tag{10}$$

We model replay as contributing a fixed effective weight $\rho > 0$. Combining it with the fresh binary-reward update from Lemma O.2 gives

$$\dot{z} = c(P)\nabla_z P - \rho\nabla_z R_{\mathcal{B}}(z), \tag{11}$$

where $c(P)$ denotes the bounded continuous coefficient from Lemma O.2.

Theorem O.4 (Replay keeps buffered modes bounded away from zero)**.** *Under Assumption O.1, suppose that from time T onward the nonempty buffer $\mathcal{B}$ is fixed and receives a fixed positive replay weight ρ. Then there exists $\epsilon > 0$ such that*

$$p_b(t) \ge \epsilon \qquad \textit{for every } b \in \mathcal{B} \textit{ and every } t \ge T.$$

Thus no buffered solution mode can vanish under the fixed-buffer mean flow.

Proof. Consider the quantity

$$\rho R_{\mathcal{B}}(z(t)) - \int_0^{P(t)} c(s)\, ds.$$

Along the flow in Equation (11),

$$\frac{d}{dt}\left[\rho R_{\mathcal{B}}(z(t)) - \int_0^{P(t)} c(s)\, ds\right] = -\left\|\rho \nabla_z R_{\mathcal{B}} - c(P)\nabla_z P\right\|_2^2 \le 0.$$

Therefore, for every $t \ge T$,

$$R_{\mathcal{B}}(z(t)) \le R_{\mathcal{B}}(z(T)) + \frac{1}{\rho}\max_{u\in[0,1]} \int_{P(T)}^{u} c(s)\, ds.$$

Define

$$C_T := R_{\mathcal{B}}(z(T)) + \frac{1}{\rho}\max_{u\in[0,1]} \int_{P(T)}^{u} c(s)\, ds.$$

Because c is bounded on $[0,1]$, $C_T < \infty$. Hence

$$R_{\mathcal{B}}(z(t)) \le C_T \qquad \text{for all } t \ge T.$$

Finally,

$$\sum_{b\in\mathcal{B}} -\log p_b(t) = kR_{\mathcal{B}}(z(t)) \le kC_T.$$

Each term in the sum is nonnegative, so $-\log p_b(t) \le kC_T$ for every $b \in \mathcal{B}$. Therefore

$$p_b(t) \ge e^{-kC_T} > 0.$$

Taking $\epsilon = e^{-kC_T}$ proves the result. □

O.4 Scope of the idealized analysis

The analysis isolates two mode-level mechanisms rather than attempting to reproduce every detail of neural optimization: binary-reward mean flow favors concentration among unequal correct modes, while fixed-buffer replay supplies a direct likelihood signal to stored modes even after they become rare. Theorem O.4 therefore applies to modes already represented in a fixed buffer under the independent-logit assumptions; discovery, changing buffer membership, and shared-parameter interactions remain empirical questions.

The experiments test the corresponding observable predictions directly. Sparse or uniform-reward rollout groups provide little or no fresh task-gradient signal (App. F.4), RLVR-only training reduces observed correct-solution breadth across scales (App. F.3), and replay preserves substantially more conditional diversity than matched controls under a fresh 64-response evaluation (App. F.1).

P Compute and Storage Overhead

P.1 Matched schedule and compute controls

Replay and its matched control share initialization, training data, optimizer settings, 3,072 optimizer updates, and $G = 16$ fresh rollouts per visited prompt. Both execute the same validation, buffer construction, scheduling, and exemplar-scoring machinery, with the replay gradient set to zero in the control. Replay therefore adds the stored-exemplar learning signal and its scoring cost without adding optimizer steps or fresh rollouts. We report the measured overhead below and separately compare against no-replay controls with a 50% larger fresh-rollout budget, allocated either to more updates or wider groups (Apps. P.3 and P.4).

P.2 STORAGE AND RUNTIME OVERHEAD

Replay adds storage for retained exemplars and computation for scoring them. Across the 75 replay runs in the main factorial, the largest estimated mean replay-buffer footprint is 444 kB. For comparison, storing a separate full reference model would require approximately 0.99–6.17 GB across the three tested model scales. These values describe component storage rather than peak accelerator memory.

We also measure runtime directly for the Qwen2.5-0.5B Level-1 control and Re:Dr runs executed on the same two A5000 nodes with matched prompts and seeds. Across 25 domain–seed pairs, replay increases learner time by 2.4% and whole-job wall-clock time by 1.1%. These end-to-end measurements include system overhead and show that replay adds modest runtime in this implementation.

The longer-horizon control below provides a substantially larger compute perturbation. Relative to the standard 8-pass control, the 12-pass control uses 44.9% more learner time and 44.7% more wall-clock time.

P.3 LONGER-HORIZON CONTROL

We test whether additional training alone closes the replay gap by extending the no-replay control from 8 to 12 passes. The longer control receives 4,608 optimizer updates instead of 3,072 and 50% more fresh rollouts, while using the same prompts, seeds, hardware, optimization settings, and evaluation procedure.

Additional training does not close either gap. Averaged across the five domains, control `pass@8` changes from .444 after 8 passes to .434 after 12 passes, compared with .748 for Re:Dr after 8 passes. Mean `PCMD` changes from .004 to .013, compared with .376 for Re:Dr. Thus the substantially longer no-replay run remains near one effective verified solution mode and remains far below replay on both correctness and conditional diversity.

`Python` makes the contrast especially clear. Only 1.5% of its control updates contain a nonzero fresh task gradient (App. F.4), and increasing the horizon reduces terminal `pass@8` from .338 to .182. At 12 passes, `Python` produces too few prompts with multiple verified responses to estimate `PCMD` under the reporting threshold.

P.4 WIDER-GROUP CONTROL

The longer-horizon control spends a 50% larger fresh-rollout budget on additional updates. We test the complementary allocation by increasing the no-replay group size from $G = 16$ to $G = 24$ while holding the training horizon fixed at 3,072 updates. This gives the wider-group control the same total number of fresh rollouts as the 12-pass control, but uses them to sample more candidates within each update rather than to perform more updates.

Wider groups do not close the replay gap. Averaged across the five domains, terminal `PCMD` is .004 for the $G = 16$ control, .008 for the $G = 24$ control, and .376 for Re:Dr. No domain-level `PCMD` change from $G = 16$ to $G = 24$ exceeds .016, and every corresponding interval includes zero. Mean `pass@8` is .444 for the $G = 16$ control and .435 for the $G = 24$ control, compared with .748 for Re:Dr. In `Python`, `pass@8` decreases from .338 to .192.

Wider groups modestly increase the fraction of updates containing both correct and incorrect responses, from .161 to .185 on the five-domain mean. The increase is small in `Python`, from .020 to .023, and largest in `MathIR`, from .351 to .442. Thus, increasing fresh sampling within each update does not substantially restore task-gradient signal in the setting where the control is most repetitive.

Together with the longer-horizon control, this experiment shows that allocating a 50% larger fresh-rollout budget either to more updates or to wider rollout groups closes neither the correctness nor the diversity gap to replay.

P.5 SINGLE-SLOT REPLAY

Re:Dr both rehearses previously verified successes and balances rehearsal across the different solution modes discovered for each prompt. To separate these effects, we train a single-slot variant that is identical to Re:Dr except that each prompt retains only its first verified solu-

tion mode. The single-slot condition therefore provides replay of a past success without the ability to balance rehearsal across multiple stored modes.

Replay of a single success accounts for a substantial part of the observed effect. Averaged across the five domains, terminal `PCMD` is .004 for the matched control, .218 for single-slot replay, and .372 for Re:Dr. Single-slot replay recovers 58% of the aggregate Re:Dr–control diversity improvement. Mean `pass@8` similarly increases from .444 for the control to .707 for single-slot replay, compared with .796 for Re:Dr. Thus, retaining and rehearsing even one verified success can preserve additional solution breadth without balancing rehearsal across multiple stored modes.

Balancing across discovered modes provides a further diversity gain. Across the 25 matched domain–seed cells, Re:Dr exceeds single-slot replay by .155 `PCMD` with a 95% interval of $[+.071, +.239]$. The additional balancing benefit is largest in `Graph` and `Countdown`, with smaller domain-specific estimates elsewhere. Together, these results distinguish two contributions of replay: retaining past successes provides a substantial benefit, while balancing rehearsal across multiple discovered solution modes provides a further aggregate diversity gain.

Part VII. Related-method comparison

Q Comparison with Diversity-Preserving RLVR Methods

Diversity-preserving RLVR methods differ in what they measure, which responses they update, and what distribution they encourage. Clip-Cov and KL-Cov (Cui et al., 2025) address token-level entropy collapse by modifying token updates or applying selective KL regularization. UCPO (Lochab et al., 2026) redistributes advantage among correct responses in the current rollout group. DMPO (Li et al., 2026b) uses group-level distribution matching between reward-derived targets and sampled responses. Verified replay instead stores previously discovered verified solution modes and rehearses one exemplar per stored mode.

Different notions of diversity. These methods do not measure interchangeable quantities. Token entropy measures uncertainty over generated tokens, while other methods evaluate diversity over sampled responses, equations, or feasible solutions. ModeBench instead defines solution-mode identity through executable verification and measures diversity among correct solution modes. Multiple response strings can therefore represent the same ModeBench solution mode, while token-level diversity can also arise from incorrect responses. Our conditional-diversity metric $\texttt{PCMD}(q) = 1 - \sum_c q_c^2$ depends only on how probability is allocated among correct solution modes. By contrast, $\texttt{pass@}K$ depends on total correctness probability, while $\texttt{distinct@}K$ depends on both correctness and the allocation among correct modes.

Fresh rollouts versus stored replay. Clip-Cov, KL-Cov, UCPO, and DMPO operate on responses available in the current rollout group. Verified replay can directly update a previously discovered solution mode even when that mode is absent from the current group because its exemplar remains in the buffer. App. O.2 formalizes this distinction in the solution-mode model: the expected on-policy signal for a mode vanishes as its probability approaches zero, while a stored replay target continues to receive a direct likelihood signal. This does not imply that fresh-rollout methods cannot preserve diversity, since shared parameters, regularization, and later samples can also affect an unsampled mode. The narrower distinction is that replay does not require rediscovery before a stored solution mode can receive another direct update.

Different targets. The methods also differ in what they attempt to balance. UCPO favors a more even allocation among correct responses represented in the current group, while DMPO matches sampled responses to a reward-derived target. Verified replay instead gives equal replay weight to the discovered solution modes represented in its buffer. Uniformity over response strings or sampled rows therefore need not imply uniformity over executable solution modes. Our replay-weighting ablation isolates this distinction within the same replay procedure: weighting stored modes by fresh discovery frequency preserves fewer verified alternatives than weighting them uniformly (App. E.8). The single-slot ablation separately shows that retaining one verified success provides a substantial benefit even without multi-mode balancing, while balancing multiple discovered modes contributes additional diversity (App. P.5).

Evaluated comparators. We evaluate UCPO and sparse RLEP-Dr directly on ModeBench at the smaller model scales (App. E.5). We also evaluate fixed Semantic-MaxEnt, GAPO, and SetPO as additional comparators (Apps. D.7 and E.6). Semantic-MaxEnt adds a frequency-based bonus to fresh verified responses, while sparse RLEP-Dr reuses verified trajectories without balancing replay across solution modes. Self-imitation learning (Oh et al., 2018) and rejection-sampling fine-tuning (Dong et al., 2023) likewise revisit successful model outputs, placing verified replay broadly in the same family. Its distinguishing feature is how stored experience is organized: one exemplar is retained per discovered solution mode and stored modes are rehearsed with equal weight.

Clip-Cov, KL-Cov, and DMPO are included as mechanism-level literature comparisons because their reported benchmarks, metrics, and training setups differ from ModeBench. The direct ModeBench experiments evaluate UCPO, RLEP-Dr, Semantic-MaxEnt, GAPO, and SetPO under the executable solution-mode framework used in this paper.

Part VIII. Data and reproducibility

R Data and Reproducibility

R.1 Evaluation pairing

Paired method comparisons match model, domain, training seed, and training endpoint. Comparisons to initialization use each method's corresponding initial evaluation. In the fresh 64-response evaluation, each training seed is paired with one initial sampling replica shared across the four methods, and all five replicas use the same initial model weights. Terminal replay effects compare trained checkpoints on matched evaluation populations. Each reported comparison states its evaluation population, sample size, and conditional-diversity eligibility rule.

R.2 Released datasets and checkpoints

The ModeBench dataset contains benchmark tasks organized by domain and difficulty level together with their documented split roles. The loading guide and evaluation guide describe how to load and evaluate the benchmark and document the role of each released split. The model repository contains released checkpoints organized by model, domain, method, and seed, together with the corresponding configurations and tokenizers. Each model entry identifies the exported version and applicable base-model license.

R.3 Analysis resources

The experiment index maps the main factorials, scale comparisons, difficulty experiments, replay-weighting ablation, and supporting controls to their corresponding runs. The analysis-input index documents the numerical inputs used in the analyses. The model-loading guide provides instructions for loading the released checkpoints. Figure inputs and rendering scripts are listed in `paper/FIGURE_MANIFEST.md`.