BEYOND TASK BALANCING: COORDINATING EXPERIENCE GROUPS IN MULTI-ENVIRONMENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-environment reinforcement learning (RL) trains language agents to retrieve information, use tools, and solve problems within a shared policy. Besides deciding how much experience each environment contributes, joint training must decide how this experience combines into each update. Group Relative Policy Optimization (GRPO) compares trajectories within each problem's group but averages groups without considering whether their policy gradients reinforce or oppose one another. We introduce Compatibility-guided Group Reweighting (CoGR), which coordinates experience groups while keeping each environment's rollout quota fixed. Within-group relative rewards decide which trajectories to reinforce, and between-group gradient compatibility decides how strongly each group contributes. CoGR scores each complete group in the current on-policy batch by its environment-balanced gradient agreement with other groups and adds an extra penalty for strongly opposing gradients. Bounded positive weights with unit mean per environment rescale only the policy-gradient loss, preserving within-group comparisons and reusing the collected trajectories. We train a 4B-parameter agent on Search (Natural Questions and HotpotQA), TextCraft, and a restricted CodeContests subset. Across two training seeds, CoGR improves environment-averaged success on the development sets by 2.39 percentage points (pp) over GRPO after 64 policy updates, including a 7.64 pp gain in Code, the weakest environment. In a single-seed comparison at the same endpoint, CoGR exceeds agent-adapted GradAlign-cosine by 5.33 pp and achieves slightly higher environment-averaged success than agent-adapted MT-GRPO with 82.51% fewer generated rollouts. These results suggest that experience-group coordination is a useful way to improve joint learning within prescribed environment allocations.

1 INTRODUCTION

General-purpose language agents must learn to retrieve information, use tools, and solve problems across heterogeneous environments (Yao et al., 2023; Liu et al., 2024; Xi et al., 2025a; Jin et al., 2025; Qian et al., 2025; Feng et al., 2025a; Wei et al., 2025). Multi-environment reinforcement learning (RL) trains these capabilities in a shared policy (Zhang et al., 2025; Xi et al., 2025b). Joint training therefore depends on two decisions, namely how much experience each environment contributes and how the learning signals in this experience combine into each update.

The second decision is especially important for Group Relative Policy Optimization (GRPO; Shao et al., 2024; DeepSeek-AI, 2025). For each problem, GRPO samples an *experience group* of trajectories and uses their relative rewards to decide which trajectories to reinforce. Yet informative reward differences within each group do not ensure compatible updates across groups. Two groups can each provide useful supervision while their policy gradients point in opposing directions. Such gradient conflicts are well documented between tasks in multi-task learning, where they can hinder joint optimization (Yu et al., 2020; Liu et al., 2021; Navon et al., 2022; Shen et al., 2024; Liu et al., 2025a). Standard GRPO averages all group losses with equal weight and does not account for these interactions.

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.

Existing methods mainly control how much each environment contributes or which problems enter training. Task-level balancing adjusts the influence of each environment or problem category through adaptive sampling (Wang et al., 2025a; Chen et al., 2025), task-wise signal normalization (Zhang et al., 2025), or learned task weights (He et al., 2024), as in Multi-Task GRPO (MT-GRPO; Ramesh et al., 2026). These controls act on an environment or category as a whole. They cannot distinguish a group whose gradient agrees with the other environments from a group in the same environment whose gradient opposes them.

Data selection acts at a finer level and chooses individual problems or samples. LIMR selects samples by their estimated learning impact (Li et al., 2025a), and gradient-guided methods choose problems by gradient alignment with a validation set in GradAlign (Yang et al., 2026) or with the training pool in LearnAlign (Li et al., 2026). With multiple environments, GradAlign's averaged reference gradient can favor environments with larger gradient norms, and LearnAlign's pool-wide average favors environments with more problems. Global selection can also shift each environment's share of training, which confounds better aggregation with changed exposure. These methods decide which problems are trained on. It remains unclear whether coordinating the groups already collected for an update can improve joint learning without changing each environment's exposure.

We study this problem with each environment's rollout quota held fixed. The allocation encodes the planned coverage of each environment and may come from a curriculum or a task-level balancer, so a method that works within it can complement task-level balancing. Fixed quotas also give GRPO, CoGR, and its ablations the same exposure to each environment, so their performance differences reflect how the update combines groups.

Our key insight is that groups with locally aligned policy gradients can reinforce one another, so gradient compatibility can guide how much each group contributes within a fixed quota. We therefore introduce **Compatibility-guided Group Reweighting (CoGR)** (Figure 1). CoGR keeps GRPO's within-group relative rewards to decide which trajectories to reinforce, and it uses between-group compatibility to decide how strongly each group contributes. It scores each complete group in the current on-policy batch by gradient agreement and adds an extra penalty for strong conflicts. This environment-balanced score averages over distinct reference problems within each environment and then equally across environments, so environments with more groups do not dominate it. Each group receives one bounded positive weight with unit mean per environment, which scales all its trajectories together and preserves within-group comparisons. Scoring reuses the collected trajectories, and only the policy-gradient loss is reweighted.

We evaluate CoGR with a 4B-parameter agent on development sets spanning retrieval, stateful tool use, and coding. Across two independent training seeds, CoGR improves environment-averaged success over GRPO by 2.39 percentage points (pp) after 64 updates from each seed's shared checkpoint, including a 7.64 pp gain in Code, the weakest environment. We also adapt MT-GRPO and GradAlign to the agent setting, with cosine scoring for GradAlign (GradAlign-cosine). In a single-seed comparison at the same update endpoint, CoGR exceeds GradAlign-cosine by 5.33 pp and achieves slightly higher environment-averaged success than MT-GRPO with 82.51% fewer generated rollouts. Matched single-seed ablations suggest that the gain depends on group-specific weight assignment and that the conflict penalty adds a further benefit at this endpoint.

Our contributions are threefold:

- **Problem formulation.** We formulate experience-group coordination within fixed environment allocations as a decision distinct from task-level balancing.
- **Method.** We develop CoGR, a bounded, environment-balanced, and conflict-sensitive group reweighting that conserves each environment's policy-gradient weight.
- **Empirical evidence.** We show improved joint learning, characterize performance–generation trade-offs against task balancing and gradient-guided selection, and test group assignment and conflict sensitivity through ablations.

2 METHOD

Overview. CoGR coordinates joint policy updates by reweighting complete on-policy experience groups within prescribed environment quotas. GRPO decides which trajectories to reinforce within

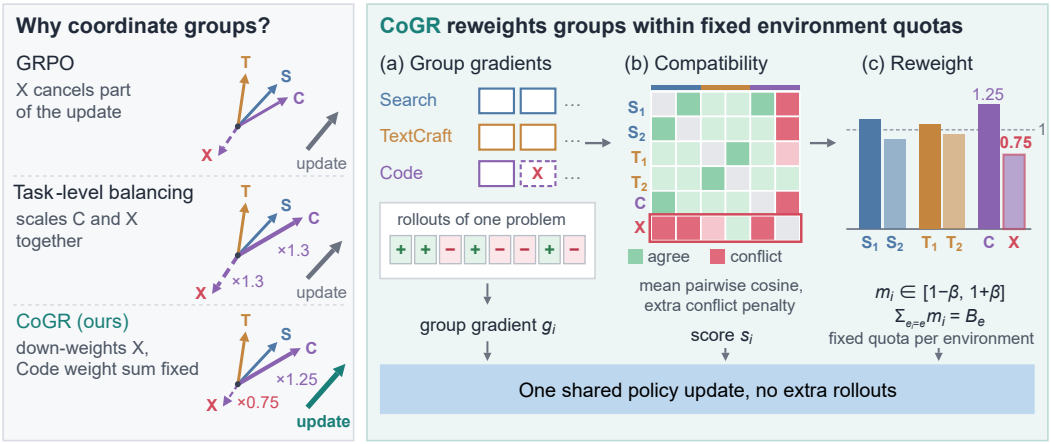

Figure 1: **Coordinating experience groups within fixed environment quotas.** In the left panel, group X in Code conflicts with the other groups. GRPO lets X cancel part of the update, and task-level balancing scales X together with the rest of Code. CoGR moves weight from X to compatible groups while the sum of the Code group weights stays fixed. In the right panel, CoGR computes a policy gradient for each complete group, scores it by transformed pairwise cosines averaged by reference problem and then equally across environments, and assigns one bounded, detached weight per group within its source environment. Scoring needs no extra rollouts. All vectors, cosines, and weights are illustrative.

each problem. CoGR then decides how much each group contributes to the shared update, using an environment-balanced and conflict-sensitive compatibility score (Figure 1).

2.1 Problem Formulation

We consider a shared language policy π_θ interacting with K environments, where θ denotes all trainable LoRA parameters (Hu et al., 2022). Environment e provides a distribution $\mathcal{D}_e$ of problems q and a terminal reward $R_e(q, \xi)$ for an interaction trajectory ξ. The learning objective is the environment-averaged reward

$$J(\theta) = \frac{1}{K} \sum_{e=1}^{K} \mathbb{E}_{q \sim \mathcal{D}_e,\, \xi \sim \pi_\theta(\cdot|q,e)} \left[R_e(q, \xi) \right]. \tag{1}$$

Training proceeds through policy-update iterations $u = 0, 1, 2, \ldots$, and θ_u denotes the parameters before update u. At update u, each environment e contributes a fixed quota of B_e experience groups collected under $\pi_{\text{old}} = \pi_{\theta_u}$, giving a batch of $B = \sum_e B_e$ groups. Group $\mathcal{E}_i$ consists of a problem with canonical identifier q_i, its environment e_i, and $G > 1$ trajectories $\{\xi_{ij}\}_{j=1}^{G}$ with rewards $\{r_{ij}\}_{j=1}^{G}$. CoGR outputs one positive group weight, or multiplier, m_i per group, and the resulting weighted loss updates θ. The multipliers must sum to B_e within each environment e, so the environment keeps its total nominal policy-gradient (PG) weight.

2.2 Within-Group Learning Signals

A group's contribution should reflect the reward comparisons that GRPO actually learns from. We therefore score each group through its complete PG loss. We retain the baseline's outcome-supervised GRPO advantages (Shao et al., 2024)

$$\bar{r}_i = \frac{1}{G} \sum_{j=1}^{G} r_{ij}, \qquad \sigma_i^2 = \frac{1}{G-1} \sum_{j=1}^{G} (r_{ij} - \bar{r}_i)^2, \qquad A_{ij} = \frac{r_{ij} - \bar{r}_i}{\sigma_i + \varepsilon_A}, \tag{2}$$

where A_{ij} is the relative advantage of trajectory j, σ_i is the sample standard deviation, and $\varepsilon_A = 10^{-6}$. Each trajectory's advantage is broadcast to its valid assistant-response tokens. Let M_{ijt} mark these tokens, and let $\ell_{ijt}^{\mathrm{PG}}(\theta)$ denote the baseline's token-level clipped PG loss (Schulman et al., 2017). The complete-group loss is

$$L_i^{\mathrm{PG}}(\theta) = \frac{1}{G}\sum_{j=1}^{G}\frac{\sum_t M_{ijt}\ell_{ijt}^{\mathrm{PG}}(\theta)}{\sum_t M_{ijt}}. \tag{3}$$

Equation (3) averages token losses within each response and then over trajectories. Tool observations provide context but receive no PG token loss. Scoring and training share the dual-clipped surrogate (Ye et al., 2020), with $\epsilon_{\mathrm{low}} = \epsilon_{\mathrm{high}} = 0.2$ (ratio-clipping interval $[0.8, 1.2]$) and a dual-clip coefficient of 3. Appendix A.1 gives the exact form of this surrogate.

2.3 Environment-Balanced Group Compatibility

We measure compatibility through group policy gradients. For each group, we define the PG gradient at the current parameters, its norm, and the pairwise cosine as

$$g_i = \nabla_\theta L_i^{\mathrm{PG}}(\theta)\big|_{\theta=\theta_u}, \qquad n_i = \|g_i\|_2, \qquad c_{ik} = \mathrm{clip}\left(\frac{\langle g_i, g_k\rangle}{n_i n_k}, -1, 1\right). \tag{4}$$

The gradients cover all trainable parameters and contain only the PG term. Cosines are computed only between valid groups, whose gradient norms are finite and exceed the numerical threshold τ in Appendix A.2.

The inner product indicates whether two groups help or hinder each other locally. By a first-order Taylor expansion, a small descent step $-\eta g_i$ changes group k's loss by $-\eta\langle g_k, g_i\rangle + O(\eta^2)$, so a positive inner product means that the step also decreases this loss. We use the cosine so that large gradient norms do not dominate the comparison, and we treat this plain-gradient argument only as a local heuristic for the AdamW update (Loshchilov & Hutter, 2019).

A mean cosine can hide strong conflicts, because different mixtures of aligned and opposing pairs can share the same mean. To give strong conflicts additional influence on the score, we transform each pairwise cosine before averaging with

$$f(c) = c - \max(-c - \delta, 0), \qquad \delta = 0.05. \tag{5}$$

The transform keeps cosines at or above $-\delta$ unchanged and subtracts the excess conflict of any pair below this margin. For candidate i, let $\mathcal{Q}_e^{(-i)}$ contain the distinct canonical identifiers of valid groups in environment e, excluding q_i. Let $\mathcal{J}_{e,q}$ index the valid groups of reference problem q. The compatibility score is

$$s_i = \frac{1}{K}\sum_{e=1}^{K}\frac{1}{|\mathcal{Q}_e^{(-i)}|}\sum_{q\in\mathcal{Q}_e^{(-i)}}\frac{1}{|\mathcal{J}_{e,q}|}\sum_{k\in\mathcal{J}_{e,q}} f(c_{ik}). \tag{6}$$

The three averages let repeated groups of one problem share that problem's weight, weight distinct problems equally within each environment, and weight all environments equally. References include the candidate's own environment but exclude every group that shares q_i. A candidate is eligible only if it has a valid gradient, a finite score, and at least four distinct valid reference problems in every environment. Ineligible groups keep unit weight, and Appendix A.2 reports eligibility rates.

2.4 Bounded Reweighting within Each Environment

Compatibility determines relative contributions only among groups from the same source environment. Let $\mathcal{I}_e$ contain the eligible groups from environment e. When $|\mathcal{I}_e| \geq 2$, we center their scores and normalize the deviations,

$$z_i = s_i - \frac{1}{|\mathcal{I}_e|}\sum_{k\in\mathcal{I}_e} s_k, \qquad d_e = \max\left\{\max_{k\in\mathcal{I}_e}|z_k|,\, \varepsilon_s\right\}, \qquad i \in \mathcal{I}_e, \tag{7}$$

where $\varepsilon_s = 0.01$ limits the amplification of small score differences. With reweighting strength $0 \le \beta < 1$, the multiplier is

$$m_i = \begin{cases} 1 + \beta z_i / d_{e_i}, & i \in \mathcal{I}_{e_i} \text{ and } |\mathcal{I}_{e_i}| \ge 2, \\ 1, & \text{otherwise.} \end{cases} \tag{8}$$

Groups above their environment's eligible-group mean receive more weight, and those below it receive less. Because the centered scores sum to zero, the multipliers in environment e still sum to B_e. Because $|z_i| \le d_{e_i}$, no multiplier moves farther than β from one. Together, these properties give $1 - \beta \le m_i \le 1 + \beta$ and $\sum_{i:e_i=e} m_i = B_e$ for every e. The default $\beta = 0.25$ gives weights in $[0.75, 1.25]$. Since $\beta < 1$, all multipliers are positive and never reverse a group's gradient direction. One multiplier scales all G trajectories of a group, which preserves their within-group relative weighting. An environment with fewer than two eligible groups, or with equal eligible scores, keeps uniform weights.

2.5 Joint Policy Update

CoGR recomputes scores and weights on each new on-policy batch and holds them fixed during the update on that batch. It minimizes

$$L_{\mathrm{CoGR}}(\theta;\theta_u) = \frac{1}{B}\sum_{i=1}^{B} \mathrm{stopgrad}(m_i) L_i^{\mathrm{PG}}(\theta) + \lambda_{\mathrm{KL}} L_{\mathrm{KL}}(\theta), \tag{9}$$

where $\mathrm{stopgrad}$ treats the multipliers as constants and L_{KL} is the baseline's regularizer against a fixed reference policy, with $\lambda_{\mathrm{KL}} = 0.001$. The KL term stays unweighted, so compatibility-based coordination acts only on reward learning. Since the multipliers sum to B_e, environment e keeps the total weight B_e/B in Equation (9). This weight equals the $1/K$ of Equation (1) under the equal quotas of all experiments, and with unequal quotas it follows the given allocation, which CoGR leaves unchanged. The accumulated gradient passes through the baseline's gradient clipping before one AdamW update.

We implement CoGR in veRL (Sheng et al., 2025) and compute pairwise inner products from distributed group-gradient shards. Scoring reuses the collected trajectories and their stored advantages, probabilities, and masks, so it adds gradient computation but no rollout collection. Algorithm 1 in Appendix A summarizes the procedure and numerical details.

3 Experiments

We evaluate joint performance, generated-rollout budgets, component contributions, environment composition, and computational cost.

3.1 Experimental Setup and Budget Accounting

Policy and environments. We train Qwen3-4B-Instruct-2507 (Yang et al., 2025) with rank-8 LoRA in veRL (Sheng et al., 2025). The three environments cover retrieval, stateful crafting, and programming. Search uses questions from Natural Questions and HotpotQA (Kwiatkowski et al., 2019; Yang et al., 2018; Jin et al., 2025; Song et al., 2025), TextCraft requires multi-step crafting (Prasad et al., 2024), and Code uses sandbox-verified problems from a restricted CodeContests subset (Li et al., 2022). Their development sets contain 48, 85, and 18 problems (Appendix B.1).

Methods and training. External comparisons branch from one shared checkpoint obtained after 32 GRPO updates. GRPO (Shao et al., 2024) and CoGR use 16 groups per environment, eight trajectories per group, FP32 precision, and a learning rate of 2×10^{-5}, and CoGR uses $\beta = 0.25$

Table 1: **CoGR has the highest final Mean and the smallest final Range in the shared-seed comparison.** Panels A1 and A2 use one shared training seed, and Panel B averages per-seed values over two seeds. Scores are development success percentages. Code is the lowest environment in every row and therefore gives Worst Env. Bold marks the best value in each block.

Method	Search ↑	TextCraft ↑	Code ↑	Mean ↑	Range (pp) ↓	Generated rollouts
Panel A1: one shared training seed, 32 policy updates						
GRPO	44.79	**44.71**	20.83	36.78	23.96	12,288
MT-GRPO	45.31	43.24	**37.50**	**42.02**	**7.81**	66,816
GradAlign-cosine	**46.88**	42.65	27.78	39.10	19.10	19,968
CoGR	44.27	43.82	25.00	37.70	19.27	12,288
Panel A2: one shared training seed, 64 policy updates						
GRPO	45.31	**47.35**	25.00	39.22	22.35	24,576
MT-GRPO	**50.52**	46.18	27.78	41.49	22.74	140,544
GradAlign-cosine	44.79	40.00	25.00	36.60	19.79	39,936
CoGR	46.35	44.71	**34.72**	**41.93**	**11.63**	24,576
Panel B: two-seed means, 64 policy updates						
GRPO	44.53	**47.06**	27.08	39.56	19.98	24,576
CoGR	**45.83**	45.29	**34.72**	**41.95**	**11.40**	24,576

(development sweep in Appendix B.10). We compare agent-adapted MT-GRPO (Ramesh et al., 2026), which uses task weighting and ratio-preserving sampling, and GradAlign-cosine (Yang et al., 2026), which selects problems by cosine similarity with a reference gradient every eight updates (Appendix B.2–B.3).

Evaluation and budgets. All three-environment models are evaluated using the same four prespecified attempts per problem, with all attempts scored and none excluded. Mean, the environment-averaged success, averages attempt successes within each problem, then equally over problems within each environment and over environments. Worst Env. is the lowest environment score, and Range is the highest minus the lowest. Generated rollouts count every trajectory generated after the shared checkpoint, including resampling and scoring trajectories, and exclude independent evaluation.

3.2 Three-Environment Performance and Learning Progress

Joint gains reach the weakest environment. Table 1 (Panel B) shows that CoGR raises Mean from 39.56% to 41.95% across two seeds. Code improves by 7.64 pp and Search by 1.30 pp, while TextCraft is 1.76 pp lower than under GRPO. Both seeds improve in Mean (+2.71 and +2.08 pp) and in Code (+9.72 and +5.56 pp). Code is the weakest environment for both methods, so Worst Env. rises from 27.08% to 34.72%, and the average within-seed Range narrows from 19.98 to 11.40 pp (Appendix B.5). TextCraft still improves under CoGR, rising from 41.18% at the shared checkpoint to 44.71% in the shared seed, although GRPO reaches 47.35%. CoGR therefore produces more balanced final environment scores, mainly by improving Code. These changes occur with identical environment quotas and nominal policy-gradient weight per environment.

Completed external comparison. At 64 updates, CoGR has the highest Mean (41.93%) and the smallest Range (11.63 pp) among the four methods (Table 1, Panel A2). Against GradAlign-cosine, CoGR improves Mean by 5.33 pp, with gains of 1.56, 4.71, and 9.72 pp in Search, TextCraft, and Code. Against MT-GRPO, CoGR improves Mean by 0.44 pp and Code by 6.94 pp, while MT-GRPO retains higher Search and TextCraft scores. At 32 updates, both external methods score higher than CoGR. Figure 2 relates this change over training to each method's generation budget.

Competitive performance with fewer generated rollouts. Figure 2(a) shows that CoGR reaches 41.93% with 24,576 generated rollouts. MT-GRPO reaches a similar 42.02% at 32 updates but generates 66,816 rollouts, so CoGR is 0.09 pp lower with 63.22% fewer generated rollouts. At 64 updates, CoGR is 0.44 pp higher than MT-GRPO with 82.51% fewer generated rollouts. In the low-budget view of Figure 2(b), CoGR exceeds GradAlign-cosine by 4.00 pp at a 1.54% smaller budget,

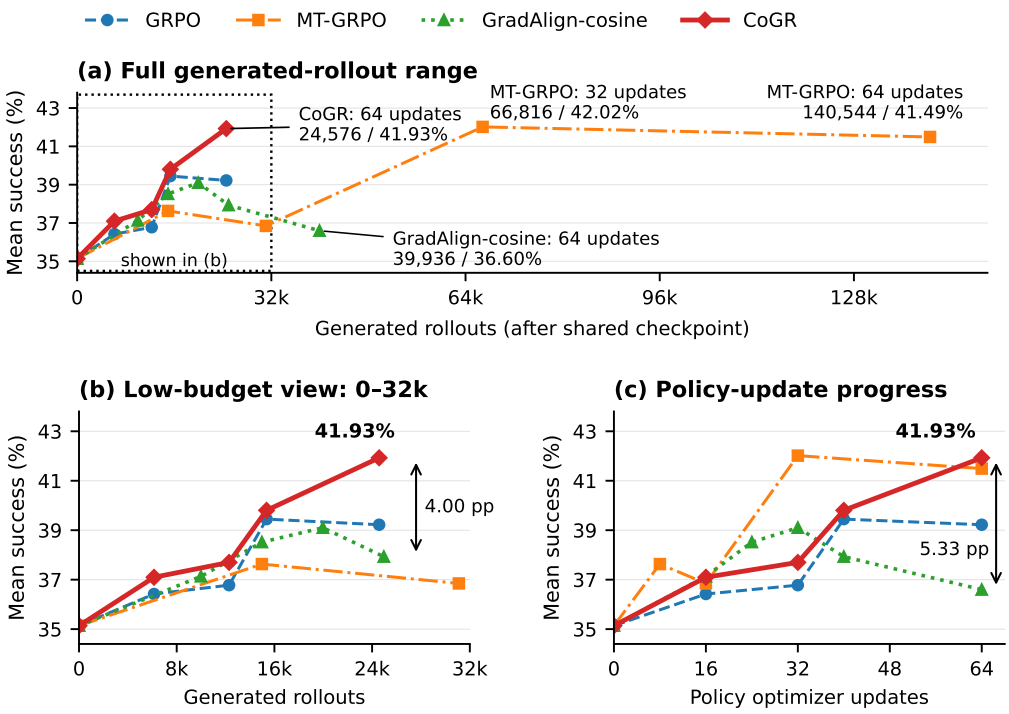

Figure 2: **Generated-rollout and policy-update views reveal distinct trade-offs.** All panels show Mean in the shared seed. (a) Mean against generated rollouts over the full range, with the dotted box marking panel (b). (b) Enlarged low-budget region. (c) Mean against policy updates. Budgets and updates count from the shared checkpoint, and lines connect measurements in order.

and GRPO reaches 39.22% at the same budget as CoGR. Appendix B.4 reports further near-budget pairs.

Early gains and final performance differ. Figure 2(c) shows that the Mean of GradAlign-cosine falls from 39.10% at 32 updates to 36.60% at 64, with all three environment scores decreasing. Over the same interval, the Mean of CoGR rises from 37.70% to 41.93%, and CoGR uses 38.46% fewer generated rollouts at the final endpoint.

3.3 Ablations: Assignment and Conflict Sensitivity

We isolate two design choices with the main three-environment seed. **CoGR-Shuffled** permutes the eligible multipliers of its own current batch within each environment, which preserves their distribution, bounds, and sum (Appendix A.4). **CoGR-Mean** replaces $f(c)$ with c, which removes the extra negative-cosine penalty and keeps all other rules (Appendix A.5).

Group-specific assignment outperforms shuffled weights. Table 2 shows that CoGR exceeds shuffled assignment by 2.27 pp in Mean at 64 updates, with the largest difference in Code (+8.33 pp). Search changes by +2.60 pp and TextCraft by −4.12 pp, and shuffled weighting remains 0.43 pp above GRPO. These results suggest that assigning each weight to its own group adds value beyond nonuniform weighting alone. A separate two-environment study gives two further positive seed-level comparisons (Appendix B.8).

Average compatibility helps, and extra conflict sensitivity adds value later. Table 2 shows that CoGR-Mean exceeds GRPO at both checkpoints (+2.49 and +1.51 pp) and leads all variants at 32 updates. At 64 updates, full CoGR is 1.19 pp higher than CoGR-Mean in Mean and 4.17 pp higher in Code. Average compatibility is therefore a useful signal on its own, and the conflict penalty adds a further benefit at the later endpoint of this run.

Table 2: **In this run, group-specific assignment outperforms shuffled weights, and extra conflict sensitivity helps at the later checkpoint.** All variants use one three-environment seed and the same quotas. The last column gives full CoGR minus the variant in pp, and bold marks the highest score in each block. No 32-update result was recorded for CoGR-Shuffled (Appendix B.7).

Updates	Variant	Search ↑	TextCraft ↑	Code ↑	Mean ↑	CoGR − variant (pp)
32	GRPO	44.79	**44.71**	20.83	36.78	+0.92
32	CoGR-Mean	**45.31**	**44.71**	**27.78**	**39.27**	−1.57
32	CoGR	44.27	43.82	25.00	37.70	0.00
64	GRPO	45.31	47.35	25.00	39.22	+2.71
64	CoGR-Shuffled	43.75	**48.82**	26.39	39.65	+2.27
64	CoGR-Mean	**46.35**	45.29	30.56	40.73	+1.19
64	CoGR	**46.35**	44.71	**34.72**	**41.93**	0.00

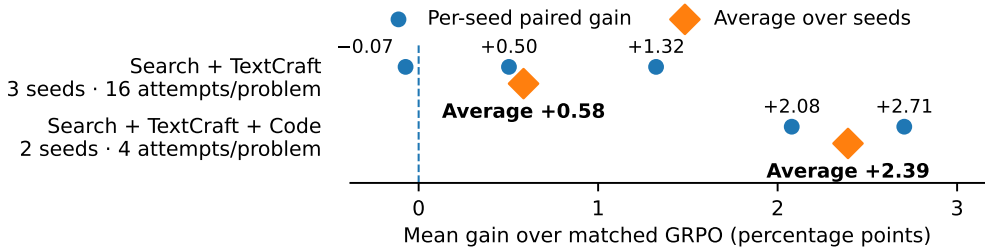

Figure 3: **Observed gains vary with environment composition.** Circles are paired Mean gains of individual training seeds after 64 updates, and diamonds average them over seeds. The two settings differ in seeds, training precision before the shared checkpoint, batch size, and evaluation settings.

3.4 Sensitivity to Environment Composition

Benefits depend on the environment combination. Figure 3 adds a Search + TextCraft setting with three training seeds. Mean rises from 43.81% to 44.39% (+0.58 pp), with +1.30 pp in TextCraft and −0.13 pp in Search, and the paired gains are +1.32, −0.07, and +0.50 pp. Code accounts for most of the larger three-environment gain, whereas TextCraft accounts for the smaller two-environment gain. Because each setting keeps its own matched baseline and evaluation protocol, Figure 3 compares observed effect sizes rather than measuring how conflict intensity changes with the environment set (Appendix B.6).

3.5 Generation and Computational Trade-offs

Generation and computation describe different trade-offs. Table 3 shows that MT-GRPO generates 115,968 trajectories beyond those accepted for its 64 updates, and GradAlign-cosine generates 15,360 scoring trajectories across eight selection rounds. CoGR scores the current update batch and needs no additional rollouts. Its cost lies instead in group-gradient computation. CoGR records 268.19 GPU·h for 64 updates versus 153.16 for GRPO, about 1.75×, and gradient capture accounts for 42.85% of the recorded update time in the first 32 updates (Appendix B.9). GradAlign-cosine records 323.45 GPU·h at 64 updates under its setup-inclusive scope, and MT-GRPO records 179.44 GPU·h when it reaches 42.02% at 32 updates.

4 Related Work

Credit assignment in agentic RL. Agentic RL assigns reward credit at different temporal scales to learn from delayed outcomes. ArCHer separates utterance-level evaluation from token-level optimization (Zhou et al., 2024), and multi-turn frameworks optimize complete agent trajectories (Wang et al., 2025b; Xi et al., 2025b). GRPO and its variants compare responses to the same problem (Shao et al., 2024; Yu et al., 2025; Liu et al., 2025b; Zheng et al., 2025), and GiGPO adds comparisons

Table 3: **Generation budgets separate update trajectories from method-specific generation.** Counts start after the shared checkpoint of the shared seed. Additional generation consists of unaccepted oversampled MT-GRPO groups or GradAlign-cosine's scoring trajectories. Superscripts mark timing scopes that differ across methods (Appendix B.9), so GPU·h values are not directly comparable.

Method	Updates	Update trajectories	Additional generated	Total generated	Recorded GPU·h
GRPO[a]	64	24,576	0	24,576	153.16
CoGR[a]	64	24,576	0	24,576	268.19
MT-GRPO[b]	32	12,288	54,528	66,816	179.44
MT-GRPO[b]	64	24,576	115,968	140,544	365.21
GradAlign-cosine[c]	40	15,360	9,600	24,960	199.55
GradAlign-cosine[c]	64	24,576	15,360	39,936	323.45

anchored at shared environment states (Feng et al., 2025b). These methods refine credit within trajectories or groups, whereas CoGR controls how strongly each complete group contributes to the shared update.

Task balancing and data selection. Multi-task and multi-domain RL regulates the exposure and signal scale of each task. Task-level controls adapt sampling over data distributions or problem categories, curricula, or domain mixtures (Wang et al., 2025a; Chen et al., 2025; Cheng et al., 2025; Li et al., 2025b), normalize signals per task (Zhang et al., 2025), or learn task weights (He et al., 2024; Ramesh et al., 2026). They set the influence of a task or category as a whole and cannot distinguish compatible from conflicting groups within one environment. Sample-level selection such as LIMR chooses training samples by their estimated learning impact (Li et al., 2025a), which decides which problems are trained on rather than how collected groups combine in an update. CoGR reweights the collected groups and keeps the quota and total policy-gradient weight of each environment.

Conflict-aware multi-task optimization. Multi-task optimizers construct shared updates from relationships between task gradients, for example through gradient projection, conflict-averse or bargaining objectives, Jacobian aggregation, smooth scalarization, and fairness-based utilities (Yu et al., 2020; Liu et al., 2021; Navon et al., 2022; Quinton & Rey, 2024; Lin et al., 2024; Ban & Ji, 2024; Liu et al., 2025a). These methods usually operate on task-level losses, whose gradients already aggregate heterogeneous groups. CoGR applies a similar geometric motivation to complete-group policy gradients before environment-level aggregation, and its bounded positive weights never reverse the direction of a group gradient.

Gradient-guided data selection. Gradient relationships also guide which problems enter training, as in LESS for instruction tuning (Xia et al., 2024) and in GradAlign and LearnAlign for RL (Yang et al., 2026; Li et al., 2026). GradAlign uses the cosine with an averaged reference gradient, which generally differs from an average of pairwise cosines. LearnAlign averages learnability-weighted pairwise cosines over the whole training pool, which favors environments with more problems. CoGR scores the current batch pairwise, averages over distinct problems and then environments, and reweights already-collected groups without separate scoring rollouts.

5 Conclusion

We presented CoGR, which keeps GRPO's within-group comparisons and uses environment-balanced gradient compatibility to coordinate experience groups within fixed environment quotas. With a 4B-parameter agent, CoGR improves Mean and the weakest environment in both three-environment training seeds. In the shared-seed comparison, it achieves the highest final Mean and the smallest Range among four methods while using fewer generated rollouts than the two agent-adapted comparators. Of the two tested settings, the one that includes Code shows the larger gain, which comes mainly from Code. These findings suggest that experience-group coordination is a useful control for joint learning. Future work will test more seeds, larger models, and broader environment sets and will reduce the group-gradient cost, the main overhead of CoGR.

REPRODUCIBILITY STATEMENT

Section 2 and Appendix A specify the CoGR objective, the scoring and reweighting rules, the numerical settings, and Algorithm 1. Appendix B.1 describes the environments and data construction, and Appendix B.2 lists the training configurations and seeds. Appendix B.3 describes the agent-adapted external methods, Appendix B.4 gives the evaluation protocol, and Appendix B.9 defines generation accounting and timing scopes. We will release the code upon acceptance.

AI USE STATEMENT

AI-assisted tools were used for proofreading and formatting.

REFERENCES

Hao Ban and Kaiyi Ji. Fair resource allocation in multi-task learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2715–2731. PMLR, 2024. URL https://proceedings.mlr.press/v235/ban24a.html.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for LLM reasoning. *arXiv preprint arXiv:2505.14970*, 2025. URL https://arxiv.org/abs/2505.14970.

Zhoujun Cheng, Shibo Hao, Tianyang Liu, Fan Zhou, Yutao Xie, Feng Yao, Yuexin Bian, Yonghao Zhuang, Nilabjo Dey, Yuheng Zha, Yi Gu, Kun Zhou, Yuqi Wang, Yuan Li, Richard Fan, Jianshu She, Chengqian Gao, Abulhair Saparov, Haonan Li, Taylor W. Killian, Mikhail Yurochkin, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Revisiting reinforcement learning for LLM reasoning from a cross-domain perspective. *arXiv preprint arXiv:2506.14965*, 2025. URL https://arxiv.org/abs/2506.14965.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. *arXiv preprint arXiv:2504.11536*, 2025a. URL https://arxiv.org/abs/2504.11536.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, 2025b. URL https://arxiv.org/abs/2505.10978.

Yifei He, Shiji Zhou, Guojun Zhang, Hyokun Yun, Yi Xu, Belinda Zeng, Trishul Chilimbi, and Han Zhao. Robust multi-task learning with excess risks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 18094–18114. PMLR, 2024. URL https://proceedings.mlr.press/v235/he24n.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. URL https://arxiv.org/abs/2503.09516.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019. doi: 10.1162/tacl_a_00276.

Shipeng Li, Zhiqin Yang, Shikun Li, Xiaobo Xia, Hengyu Liu, Xinghua Zhang, Gaode Chen, Dong Fang, Ying Tai, and Zhe Peng. LearnAlign: Data selection for LLM reinforcement learning with improved gradient alignment. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://arxiv.org/abs/2506.11480.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. LIMR: Less is more for RL scaling. *arXiv preprint arXiv:2502.11886*, 2025a. URL https://arxiv.org/abs/2502.11886.

Yu Li, Zhuoshi Pan, Honglin Lin, Mengyuan Sun, Conghui He, and Lijun Wu. Can one domain help others? A data-centric study on multi-domain reasoning via reinforcement learning. *arXiv preprint arXiv:2507.17512*, 2025b. URL https://arxiv.org/abs/2507.17512.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Xi Lin, Xiaoyuan Zhang, Zhiyuan Yang, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Smooth Tchebycheff scalarization for multi-objective optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 30479–30509. PMLR, 2024. URL https://proceedings.mlr.press/v235/lin24y.html.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2110.14048.

Qiang Liu, Mengyu Chu, and Nils Thuerey. ConFIG: Towards conflict-free training of physics informed neural networks. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2408.11104.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

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. *arXiv preprint arXiv:2503.20783*, 2025b. URL https://arxiv.org/abs/2503.20783.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. URL https://arxiv.org/abs/2202.01017.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024. doi: 10.18653/v1/2024.findings-naacl.264.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025. URL https://arxiv.org/abs/2504.13958.

Pierre Quinton and Valérian Rey. Jacobian descent for multi-objective optimization. *arXiv preprint arXiv:2406.16232*, 2024. URL https://arxiv.org/abs/2406.16232.

Shyam Sundhar Ramesh, Xiaotong Ji, Matthieu Zimmer, Sangwoong Yoon, Zhiyong Wang, Haitham Bou Ammar, Aurelien Lucchi, and Ilija Bogunovic. Multi-task GRPO: Reliable LLM reasoning across tasks. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.05547.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

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

Jiayi Shen, Qi Wang, Zehao Xiao, Nanne Van Noord, and Marcel Worring. GO4Align: Group optimization for multi-task alignment. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c98987c5ec4f30920d7190dc699e3daf-Paper-Conference.pdf.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *European Conference on Computer Systems*, 2025. URL https://arxiv.org/abs/2409.19256.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-Searcher: Incentivizing the search capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025. URL https://arxiv.org/abs/2503.05592.

Zhenting Wang, Guofeng Cui, Yu-Jhe Li, Kun Wan, and Wentian Zhao. DUMP: Automated distribution-level curriculum learning for RL-based LLM post-training. *arXiv preprint arXiv:2504.09710*, 2025a. URL https://arxiv.org/abs/2504.09710.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b. URL https://arxiv.org/abs/2504.20073.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7909–7928, 2025. doi: 10.18653/v1/2025.emnlp-main.401.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, 2025a. doi: 10.18653/v1/2025.acl-long.1355.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. AgentGym-RL: Training LLM agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*, 2025b. URL https://arxiv.org/abs/2509.08755.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54104–54132. PMLR, 2024. URL https://proceedings.mlr.press/v235/xia24c.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Ningyuan Yang, Weihua Du, Weiwei Sun, Sean Welleck, and Yiming Yang. GradAlign: Gradient-aligned data selection for LLM reinforcement learning. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2602.21492.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in MOBA games with deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1912.09729.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2001.06782.

Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Hongning Wang, Jie Tang, and Yuxiao Dong. AgentRL: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *arXiv preprint arXiv:2510.04206*, 2025. URL https://arxiv.org/abs/2510.04206.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025. URL https://arxiv.org/abs/2507.18071.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24t.html.

A CoGR Algorithm and Implementation Details

This appendix accompanies Section 2. Equation numbers (1)–(9) refer to the method section, and appendix equations use the prefix A. All gradients are taken with respect to the trainable LoRA parameters, with the backbone weights frozen.

A.1 Native GRPO Loss

CoGR retains outcome-supervised GRPO (Shao et al., 2024) and the baseline's dual-clipped PG surrogate (Ye et al., 2020), implemented in veRL v0.5.0 (Sheng et al., 2025). For a sampled assistant token a_{ijt} with causal context h_{ijt}, the numerically bounded importance ratio is

$$\rho_{ijt}(\theta) = \exp(\mathrm{clip}[\log \pi_\theta(a_{ijt} \mid h_{ijt}) - \log \pi_{\mathrm{old}}(a_{ijt} \mid h_{ijt}), -20, 20]) \,. \tag{A1}$$

The old log probabilities are stored with the batch and held fixed. With ratio-clipping widths $\epsilon_{\mathrm{low}} = \epsilon_{\mathrm{high}} = 0.2$, the token-level PG loss is

$$\begin{aligned} b_{ijt}(\theta) &= \max\big\{-A_{ij}\rho_{ijt}(\theta),\ -A_{ij}\,\mathrm{clip}\big(\rho_{ijt}(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}}\big)\big\}\,, \\ \ell_{ijt}^{\mathrm{PG}}(\theta) &= \begin{cases} \min\{-c_{\mathrm{dual}}A_{ij},\ b_{ijt}(\theta)\}, & A_{ij} < 0, \\ b_{ijt}(\theta), & A_{ij} \geq 0, \end{cases} \qquad c_{\mathrm{dual}} = 3. \end{aligned} \tag{A2}$$

The same surrogate is used in gradient capture and training. The group loss follows Equation (3), so masked token means are computed for each response and then averaged over the group's trajectories. Tool observations remain in the causal context but have zero mask entries. A group whose rewards are all equal has zero advantages under Equation (2) and therefore a zero PG gradient. The KL term is veRL's low-variance sampled estimator against the fixed reference policy with $\lambda_{\mathrm{KL}} = 0.001$, averaged over response tokens and trajectories. CoGR applies m_i only to the PG term, rewards exclude KL, and the entropy coefficient is zero.

A.2 Gradient Validity and Reference Construction

Let $H_{ik} = \langle g_i, g_k \rangle$ be the Gram matrix of group gradients, and let $n_i = \sqrt{\max(H_{ii}, 0)}$. The validity threshold and the set of valid groups are

$$\begin{aligned} \mathcal{N}_+ &= \{n_i : n_i > 0 \text{ and } n_i \text{ is finite}\}, \\ \tau &= \begin{cases} \max\{10^{-12},\ 10^{-8}\,\mathrm{median}(\mathcal{N}_+)\}, & \mathcal{N}_+ \neq \varnothing, \\ 10^{-12}, & \mathcal{N}_+ = \varnothing, \end{cases} \\ \mathcal{V} &= \{i : n_i > \tau \text{ and } n_i \text{ is finite}\}. \end{aligned} \tag{A3}$$

Cosines are computed only between valid groups. The reference sets used by Equation (6) are

$$\mathcal{Q}_e^{(-i)} = \{q_k : k \in \mathcal{V},\ e_k = e,\ q_k \neq q_i\}, \qquad \mathcal{J}_{e,q} = \{k \in \mathcal{V} : e_k = e,\ q_k = q\}. \tag{A4}$$

Every group that shares the candidate's canonical problem identifier is therefore excluded. A candidate is eligible if it is valid, has at least four distinct reference problems in every environment, and has a finite score. A group can serve as a reference without being an eligible candidate. Environments with fewer than two eligible candidates keep all multipliers at one, and ineligible groups always keep unit weight, so the bound and sum identities in Section 2.4 hold for every group. Table A2 lists all settings.

Reweighting coverage. Table A1 reports how many groups entered reweighting in updates 1–32 of training seed `2026091271`, which cover 512 groups per environment. A group with nonzero PG has a PG gradient that is not identically zero, which requires unequal rewards within the group. Eligible groups additionally satisfy the conditions above, and only eligible groups can receive multipliers other than one. Code has the largest eligible share, which is consistent with Code receiving most of the gain in Table 1.

Table A1: **Groups entering reweighting.** Training seed `2026091271`, updates 1–32, and 512 groups per environment.

Environment	Nonzero-PG groups	Eligible groups
Search	159 (31.05%)	117 (22.85%)
TextCraft	193 (37.70%)	136 (26.56%)
Code	295 (57.62%)	196 (38.28%)

Table A2: **Training and CoGR settings.** The ratio-clipping widths belong to the GRPO surrogate and are separate from the reweighting strength β. †Main experiments. Table B2 lists the learning rate of each experiment block.

Quantity	Setting	Quantity	Setting
Trajectories per group G	8	LoRA rank / alpha	8 / 8, all linear layers
Groups per environment B_e	16	Policy learning rate†	2×10^{-5}
Reweighting strength β	0.25	AdamW moments	(0.9, 0.999)
Negative-cosine margin δ	0.05	Weight decay	0.01
Minimum references per environment	4	Gradient clipping	1.0
Score normalization floor ε_s	0.01	KL coefficient λ_{KL}	0.001
Advantage constant ε_A	10^{-6}	Ratio clipping $\epsilon_{\mathrm{low}}/\epsilon_{\mathrm{high}}$	0.2 / 0.2
Norm thresholds (absolute / relative)	10^{-12} / 10^{-8}	Dual-clip coefficient	3
Optimizer steps per batch	1	Entropy coefficient	0

A.3 Execution and Distributed Gradient Capture

Training uses FSDP2 in veRL. Each rank forms a local Gram matrix from its shards of the group gradients, corrects replicated dimensions for their replication factor, and a SUM all-reduce yields the global Gram matrix in float64. Capture uses the same old log probabilities, advantages, masks, and loss normalization as the subsequent update. It performs no optimizer or scheduler step, restores the random-number-generator state and model mode, and clears temporary gradients. Algorithm 1 summarizes one update.

Algorithm 1. One CoGR policy update.

```
Input: policy parameters theta_u; optimizer and scheduler states;
       fixed KL reference pi_ref; environment samplers with quotas {B_e};
       group size G; reweighting strength beta.
Output: updated parameters theta_(u+1) and optimizer/scheduler states.

 1  Draw B_e problems from each environment e under the prescribed
 ↪  schedule.
 2  Generate G on-policy trajectories per problem; store terminal
 ↪  rewards,
    old log probabilities, canonical problem IDs, and response masks.
 3  Compute within-group advantages using Equation (2).
 4  Save random-number-generator states and model modes.
 5  Capture complete-group PG-only gradients at theta_u using Equation (3)
    and form their global Gram matrix from distributed gradient shards.
 6  Restore saved states and modes; clear temporary capture gradients.
 7  Compute gradient validity and reference sets using Equations
 ↪  (A3)--(A4).
 8  Initialize m_i <- 1 for every group and I_e <- empty for every
 ↪  environment.
 9  For each valid candidate group i:
      If every environment supplies at least four distinct valid
      ↪  references:
        Compute s_i using Equations (4)--(6).
        If all required score quantities are finite, add i to I_(e_i).
10  For each environment e with at least two eligible groups:
      Center and normalize their scores using Equation (7).
      Assign multipliers using Equation (8).
11  Detach all multipliers and attach m_i to every trajectory in group i.
12  Accumulate the PG-weighted, KL-unweighted loss in Equation (9).
13  Apply gradient clipping, perform one AdamW step,
    and advance the learning-rate scheduler once.
14  Return theta_(u+1) and the updated optimizer/scheduler states.
```

Scoring and the weighted update use the same freshly collected trajectories, so CoGR generates no separate candidate or reference rollouts.

A.4 Weight-Shuffling Control

CoGR-Shuffled tests whether the assignment of multipliers to groups matters (Table 2). At each update, the shuffled branch computes CoGR multipliers from its own on-policy batch, samples a random permutation ϖ_e of the eligible groups within each environment, and uses

$$m_i^{\text{shuf}} = \begin{cases} m_{\varpi_{e_i}(i)}, & i \in \mathcal{I}_{e_i}, \\ 1, & \text{otherwise.} \end{cases} \tag{A5}$$

The permutation preserves the multiplier distribution, bounds, and environment-wise sums while randomizing which group receives which weight. Permutations use an independent random-number generator.

A.5 Mean-Only Compatibility Ablation

CoGR-Mean isolates the extra conflict penalty in Equation (5) (Table 2). It uses the raw pairwise cosine, giving

$$s_i^{\text{mean}} = \frac{1}{K} \sum_{e=1}^{K} \frac{1}{|\mathcal{Q}_e^{(-i)}|} \sum_{q \in \mathcal{Q}_e^{(-i)}} \frac{1}{|\mathcal{J}_{e,q}|} \sum_{k \in \mathcal{J}_{e,q}} c_{ik}. \tag{A6}$$

By linearity of the nested averages, the full score in Equation (6) decomposes exactly as

$$s_i = s_i^{\text{mean}} - p_i, \qquad p_i = \frac{1}{K}\sum_{e=1}^{K} \frac{1}{|\mathcal{Q}_e^{(-i)}|} \sum_{q \in \mathcal{Q}_e^{(-i)}} \frac{1}{|\mathcal{J}_{e,q}|} \sum_{k \in \mathcal{J}_{e,q}} \max(-c_{ik} - \delta, 0) \geq 0, \quad \text{(A7)}$$

where p_i is the environment-balanced average conflict beyond the margin. CoGR-Mean keeps all other rules, including reference construction, eligibility, centering and normalization in Equation (7), $\beta = 0.25$, and the unweighted KL term. In Algorithm 1, only step 9 uses Equation (A6) instead of Equation (6).

B Experimental Protocol, Checkpoints, and Detailed Results

This appendix accompanies Section 3. Update counts and generated-rollout budgets start after the shared 32-update GRPO checkpoint unless stated otherwise. Success rates are percentages, differences are in percentage points (pp), and all results are development-set evaluations.

B.1 Environments and Data Construction

Table B1: **Training and development coverage.** The two-environment setting uses Search and TextCraft, and the three-environment setting adds Code.

Environment	Train	Dev	Interaction and binary outcome
Search	64	48	Real retrieval, answer success under substring exact match.
TextCraft	256	85	Stateful multi-turn tool interaction, successful terminal crafting.
Code	666	18	Restricted Codeforces problems from CodeContests, execution-based success in a sandbox.

Search. Questions come from Natural Questions and HotpotQA. The sampler keeps a cumulative 29:35 allocation between the two sources, cycles through seed-shuffled pools, and shares its problem sequence across matched branches, while trajectories remain on-policy for each branch. Questions are deduplicated by normalized text.

TextCraft. Tasks require multi-turn tool interaction with a persistent environment state. The binary reward records successful terminal crafting rather than intermediate progress. The two-environment development set is the deduplicated union of two development subsets.

Code. We use the `train` and `valid` splits of `deepmind/code_contests` (revision `802411c3010cb00d1b05bad57ca77365a3c699d6`) and keep Codeforces problems rated 800–1200 with prompts of at most 2,048 tokens and a single integer or yes/no output per test case. Reference solutions are used only to validate the execution pipeline and never appear in prompts or training. Training and development problems have no overlap in problem IDs, description hashes, or structure hashes. Each test has limits of 3 CPU seconds and 6 wall seconds, each program has limits of 120 CPU seconds and 180 wall seconds, and each sandbox has 1 GiB of memory.

B.2 Training Configurations and Run Identity

The base policy is Qwen3-4B-Instruct-2507 with rank-8 LoRA, trained in veRL v0.5.0 with FSDP2 and vLLM generation. Table A2 lists the optimizer, loss, and CoGR settings, and Table B2 lists the experiment blocks. GRPO, CoGR, and both ablations generate 16 groups per environment with eight trajectories each, so a three-environment update generates 384 trajectories and a two-environment update generates 256. CoGR multipliers lie in $[0.75, 1.25]$ and sum to 16 within each environment.

Table B2: **Experiment blocks supporting Section 3.** Evaluation attempts are per problem and do not increase the number of training seeds.

Experiment block	Seeds	Precision (prefix / branch)	Policy LR	Updates (attempts)
GRPO and CoGR, three environments	2	FP32 / FP32	2×10^{-5}	16, 32, 40, 64 (4)
MT-GRPO, agent-adapted	1	Shared prefix	2×10^{-5}	8, 16, 32, 64 (4)
GradAlign-cosine, agent-adapted	1	Shared prefix	2×10^{-5}	16, 24, 32, 40, 64 (4)
CoGR-Mean, three environments	1	FP32 / FP32	2×10^{-5}	32, 64 (4)
CoGR-Shuffled, three environments	1	FP32 / FP32	2×10^{-5}	64 (4)
Search–TextCraft composition	3	BF16 / FP32	2×10^{-5}	32 (4), 64 (16)
Two-environment assignment study	2	FP32	10^{-5}	32, 64 (4)

The 16- and 40-update GRPO and CoGR evaluations use the shared seed only. The three-environment training seeds are `2026091271` and `2026091671`. All external comparisons and both ablations branch from the shared checkpoint of the first seed, and the shuffled branch uses the independent permutation seed `2026091771`. The composition seeds are `2026091211`–`2026091213`, and the two development seeds of the assignment study are `2026090801` and `2026090802`.

Decoding and system configuration. Three-environment training uses at most 2,048 prompt tokens, 4,096 response tokens, and 6,144 total tokens, with temperature 1.3, top-p 1.0, and thinking disabled. Trajectories have at most 12 assistant turns and one tool call per turn. Rollouts are asynchronous with tensor parallelism 1 and four agent workers on four GPUs, namely two actor ranks, one rollout GPU, and one Search-service GPU. The shared 32-update prefix costs 76.40 GPU-hours and is excluded from all method costs. Appendix B.4 gives the evaluation configuration.

B.3 Agent-Adapted External Methods

Both external methods run in the same training stack, start from the same shared checkpoint, and use the same environments and evaluation as CoGR. They are agent-adapted versions rather than reproductions of every configuration in the original papers.

MT-GRPO. The task-logit vector starts at zero, giving equal initial probabilities. For each task, the controller combines the surrogate-loss improvement of the policy update on the current batch with 0.2 times an exponential moving average (EMA) of performance before filtering. The task logits are optimized with Adam (learning rate 0.025, weight decay 10^{-5}, moments (0.9, 0.999), epsilon 10^{-8}), and the performance and filter-rate EMAs use 0.5 and 0.9. The sampler targets 48 accepted groups of eight trajectories, drawing task counts approximately from a multinomial distribution with the current task probabilities. It oversamples by a factor of 3, proposes at most 144 groups per round, and allows up to two additional resampling rounds. Only accepted groups enter the policy update, while all generated proposals count toward the generation budget. A group is accepted only if its eight rewards are finite and not all equal, so all-failure and all-success groups are rejected. The cap of 5 applies to the filter-rate compensation multiplier $\min\{1/\operatorname{clip}(1-\rho, 0.01, 1), 5\}$, where ρ is the filter-rate EMA, and is not an acceptance threshold. Accepted groups are first trimmed to each environment's target count and then fill the remaining slots, and rejected groups are never used. If fewer than 48 groups are accepted after the last round, the update uses the smaller batch, and no update is made if none is accepted. In the reported run, all 64 updates used 48 groups, with six requiring one proposal round and 58 requiring two.

GradAlign-cosine. Selection is refreshed every eight policy updates, starting before the first update. Each refresh scores 144 candidate problems (48 per environment) with eight trajectories each against 48 reference problems (16 per environment) with 16 trajectories each, which generates 1,920 scoring trajectories. Each candidate is scored by the cosine between its PG-only LoRA gradient and the mean reference gradient, and the global top 48 problems are used for the next eight GRPO updates with fresh on-policy trajectories. Scoring trajectories are not reused for updates. Because the development sets are reserved for evaluation, the reference problems come from the deduplicated training pool rather than a validation set. They are fixed across rounds and chosen by a prespecified hash ordering, and candidates come from the remaining 48 Search, 240 TextCraft, and 650 Code training problems. Neither pool shares problem identifiers or exact prompts with the development

sets. Reference rollouts and gradients are regenerated under the current policy at every refresh. Candidates are distinct within a round but may recur across rounds. Zero reference gradients remain in the mean, a zero-gradient candidate receives score zero and stays eligible, and ties are broken by a fixed hash ordering.

B.4 Evaluation and the Complete External-Comparison Curve

For each problem, binary successes are averaged over evaluation attempts. Problems are weighted equally within each environment, and environments are weighted equally in Mean. Repeated attempts therefore estimate single-attempt success rather than pass-at-k. Every reported three-environment checkpoint has a binary score for all 604 prespecified attempts, comprising 192 Search, 340 TextCraft, and 72 Code episodes. No problems or attempt keys are excluded, and cross-seed summaries weight training seeds equally. Table B4 lists every checkpoint in Figure 2, and Table B5 lists the near-budget comparisons.

Decoding. All evaluations use synchronous Hugging Face generation with eager attention, temperature 1.0, top-p 1.0, thinking disabled, and at most one tool call per assistant turn. Table B3 lists the token and turn budgets. The response window counts assistant, tool, and template tokens over the whole episode and is not renewed at each turn. The three-environment evaluator runs in FP32 with top-k disabled. The archived two-environment evaluator loads the base model in BF16 and inherits top-k from the model's generation configuration, whose value was not recorded. Decoding is therefore matched within each setting but may differ across settings.

Table B3: **Evaluation budgets in tokenizer tokens.** Search and TextCraft use the same budgets in the two- and three-environment settings.

Environment	Settings	Prompt	Per-turn output	Response window	Assistant / tool turns
Search	Both	1,024	256	256	2 / 1
TextCraft	Both	1,024	1,536	1,536	12 / 11
Code	Three-env.	2,048	4,096	4,096	1 / 0

Termination and scoring. An episode ends at EOS, at the response window or turn limit, or when no tool call can be parsed or a tool call fails. No custom stop string is used. Search takes the last tagged answer, or the final assistant text if no tag is present, and applies normalized answer and alias matching. TextCraft success is the terminal crafting reward of the persistent environment. In Code, compilation errors, runtime errors, wrong answers, and timeouts count as failures. In three-environment evaluation, malformed tool calls and unknown tool names end the interaction and are scored from the resulting answer or environment state, and ordinary invalid-action responses remain part of the episode. Tool-execution exceptions, Search service failures, and Code grading transport or receipt errors leave an attempt incomplete rather than assigning a model failure. Interrupted evaluations resume with the original attempt keys, initial states, and generation seeds, and previously scored attempts, including task failures, are retained. Aggregation requires exactly one binary score for each of the 604 registered attempts and rejects missing or duplicate keys.

Paired attempt seeds. Each attempt seed is derived by hashing the protocol identifier, environment, problem identifier, initial-state hash, evaluation seed, and attempt index. The method, checkpoint, and training seed do not enter the hash, so all compared models face the same attempt seeds for each problem. The three-environment protocol uses the evaluation-seed and attempt-index pairs (2026091273, 0), (2026091273, 1), (2026091274, 0), and (2026091274, 1). The two-environment composition study uses evaluation seed 2026091241 with attempt indices 0–3 for four attempts. Its 16-attempt endpoints and the longer-horizon continuation use evaluation seeds 2026091241–2026091244 with indices 0–3. The assignment study uses a separate schedule with evaluation seeds 2026090821 and 2026090822.

Table B4: **All measured checkpoints in Figure 2.** Shared training seed and four attempts per problem.

Updates	Method	Search	TextCraft	Code	Mean	Generated rollouts
0	Shared checkpoint	40.6250	41.1765	23.6111	35.1375	0
16	GRPO	42.7083	42.9412	23.6111	36.4202	6,144
32	GRPO	44.7917	44.7059	20.8333	36.7770	12,288
40	GRPO	46.8750	46.4706	25.0000	39.4485	15,360
64	GRPO	45.3125	47.3529	25.0000	39.2218	24,576
16	CoGR	41.6667	43.2353	26.3889	37.0969	6,144
32	CoGR	44.2708	43.8235	25.0000	37.6981	12,288
40	CoGR	45.8333	44.4118	29.1667	39.8039	15,360
64	CoGR	46.3542	44.7059	34.7222	41.9274	24,576
8	MT-GRPO	41.6667	42.0588	29.1667	37.6307	14,976
16	MT-GRPO	44.7917	43.5294	22.2222	36.8478	31,104
32	MT-GRPO	45.3125	43.2353	37.5000	42.0159	66,816
64	MT-GRPO	50.5208	46.1765	27.7778	41.4917	140,544
16	GradAlign-cosine	43.7500	42.6471	25.0000	37.1324	9,984
24	GradAlign-cosine	47.3958	41.7647	26.3889	38.5165	14,976
32	GradAlign-cosine	46.8750	42.6471	27.7778	39.0999	19,968
40	GradAlign-cosine	46.3542	43.8235	23.6111	37.9296	24,960
64	GradAlign-cosine	44.7917	40.0000	25.0000	36.5972	39,936

Table B5: **Near-budget comparisons between observed checkpoints.** Budget is the comparator's rollouts relative to CoGR's, the next two columns give Mean, and positive ΔMean favors CoGR.

CoGR (upd. / rollouts)	Comparator (upd. / rollouts)	Budget	CoGR	Comp.	ΔMean
32 / 12,288	MT-GRPO 8 / 14,976	+21.875%	37.6981	37.6307	+0.0674
32 / 12,288	GradAlign-cosine 16 / 9,984	−18.750%	37.6981	37.1324	+0.5658
32 / 12,288	GradAlign-cosine 24 / 14,976	+21.875%	37.6981	38.5165	−0.8184
64 / 24,576	MT-GRPO 16 / 31,104	+26.5625%	41.9274	36.8478	+5.0796
64 / 24,576	GradAlign-cosine 40 / 24,960	+1.5625%	41.9274	37.9296	+3.9978

B.5 Independent Three-Environment Replication and Environment Balance

Table B6 gives the per-seed results behind Table 1B and Figure 3. At 64 updates, the two-seed Mean gain is +2.39 pp, with +7.64 pp in Code, +1.30 pp in Search, and −1.76 pp in TextCraft, and the corresponding gain at 32 updates is +0.61 pp. Both seeds have a higher Worst Env. and a lower Range under CoGR at both endpoints. With two seeds, we report per-seed results rather than significance tests.

Table B6: **Per-seed three-environment results.** Worst Env. is the lowest environment success rate, and Range is the highest minus the lowest.

Seed	Updates	Method	Search	TextCraft	Code	Mean	Worst Env.	Range
1	32	GRPO	44.7917	44.7059	20.8333	36.7770	20.8333	23.9583
1	32	CoGR	44.2708	43.8235	25.0000	37.6981	25.0000	19.2708
1	64	GRPO	45.3125	47.3529	25.0000	39.2218	25.0000	22.3529
1	64	CoGR	46.3542	44.7059	34.7222	41.9274	34.7222	11.6319
2	32	GRPO	44.7917	46.4706	27.7778	39.6800	27.7778	18.6928
2	32	CoGR	46.3542	44.4118	29.1667	39.9775	29.1667	17.1875
2	64	GRPO	43.7500	46.7647	29.1667	39.8938	29.1667	17.5980
2	64	CoGR	45.3125	45.8824	34.7222	41.9724	34.7222	11.1601

B.6 Environment Composition and Three-Seed Two-Environment Results

Table B7 lists all Search–TextCraft checkpoints. Across the three seeds, the paired 64-update differences average −0.13 pp in Search, +1.30 pp in TextCraft, and +0.58 pp in Mean, with sample standard deviations of 1.71, 3.09, and 0.70 pp. The two- and three-environment settings differ in seeds, prefix precision, per-update group count, evaluation attempts, and evaluation decoding (Appendix B.4), so Figure 3 compares paired effects within each setting rather than measuring how conflict changes with the number of environments.

Table B7: **All Search–TextCraft checkpoints.** Intermediate checkpoints use four attempts per problem and endpoints use 16.

Seed	Updates	Attempts	Method	Search	TextCraft	Mean
1	32	4	GRPO	42.7083	41.4706	42.0895
1	32	4	CoGR	42.7083	46.7647	44.7365
1	64	16	GRPO	45.1823	42.7206	43.9514
1	64	16	CoGR	43.4896	47.0588	45.2742
2	32	4	GRPO	42.1875	40.2941	41.2408
2	32	4	CoGR	40.1042	40.0000	40.0521
2	64	16	GRPO	40.3646	45.3676	42.8661
2	64	16	CoGR	42.0573	43.5294	42.7934
3	32	4	GRPO	44.2708	44.4118	44.3413
3	32	4	CoGR	42.7083	43.2353	42.9718
3	64	16	GRPO	44.6615	44.5588	44.6101
3	64	16	CoGR	44.2708	45.9559	45.1134

B.7 Three-Environment Component Ablations

Table B8 lists all ablation checkpoints. At 32 updates CoGR-Mean exceeds CoGR by 1.57 pp in Mean, whereas at 64 updates CoGR exceeds CoGR-Mean by 1.19 pp. The two ablation gaps are separate comparisons and do not decompose the total effect of CoGR.

Table B8: **All component-ablation checkpoints.** Shared three-environment seed, identical environment quotas, eight trajectories per group, and four attempts per problem.

Updates	Method	Search	TextCraft	Code	Mean	Generated rollouts
32	GRPO	44.7917	44.7059	20.8333	36.7770	12,288
32	CoGR-Mean	45.3125	44.7059	27.7778	39.2654	12,288
32	CoGR	44.2708	43.8235	25.0000	37.6981	12,288
64	GRPO	45.3125	47.3529	25.0000	39.2218	24,576
64	CoGR-Shuffled	43.7500	48.8235	26.3889	39.6541	24,576
64	CoGR-Mean	46.3542	45.2941	30.5556	40.7346	24,576
64	CoGR	46.3542	44.7059	34.7222	41.9274	24,576

B.8 Two-Environment Assignment Study

Table B9 reports a separate Search–TextCraft study with two development seeds. CoGR exceeds CoGR-Shuffled in Mean by +1.63 and +0.44 pp at 32 updates, and by +0.47 pp at 64 updates for the seed with a completed shuffled branch. These results are reported separately from the three-environment ablation because the task composition, learning rate, and seeds differ.

Table B9: **Two-environment assignment study.** Two development seeds, learning rate 10^{-5}, FP32 branches, and four attempts per problem.

Seed	Updates	Method	Search	TextCraft	Mean
1	32	GRPO	41.6667	36.7647	39.2157
1	32	CoGR	42.1875	42.6471	42.4173
1	32	CoGR-Shuffled	40.1042	41.4706	40.7874
1	64	GRPO	37.5000	43.5294	40.5147
1	64	CoGR	40.6250	42.9412	41.7831
1	64	CoGR-Shuffled	41.1458	41.4706	41.3082
2	32	GRPO	39.5833	42.3529	40.9681
2	32	CoGR	40.6250	41.1765	40.9007
2	32	CoGR-Shuffled	40.6250	40.2941	40.4596
2	64	GRPO	41.6667	41.7647	41.7157
2	64	CoGR	42.1875	41.4706	41.8290

B.9 Generation Accounting and Timing Scopes

A generated rollout is one complete trajectory generated for training or for method-specific selection. The count includes MT-GRPO's proposals, resampled groups, and discarded trajectories, as well as GradAlign-cosine's candidate and reference scoring. Reusing a scored trajectory for the update does not count as a second generation. CoGR computes gradients on existing trajectories and generates nothing extra. Evaluation and the shared prefix are excluded, and rollout counts do not measure tokens, tool calls, or GPU time.

Table B10: **Complete cost record of the external comparison.** A dash marks a checkpoint with a cost record but no evaluation. All runs use four training GPUs, and letters give the timing scope.

Method (scope)	Updates	Generated rollouts	Mean (%)	GPU·h
GRPO (a)	64	24,576	39.2218	153.16
CoGR (a)	32	12,288	37.6981	133.93
CoGR (a)	64	24,576	41.9274	268.19
CoGR-Mean (a)	64	24,576	40.7346	267.44
MT-GRPO (b)	8	14,976	37.6307	41.07
MT-GRPO (b)	16	31,104	36.8478	87.72
MT-GRPO (b)	24	48,384	—	134.52
MT-GRPO (b)	32	66,816	42.0159	179.44
MT-GRPO (b)	40	85,248	—	224.96
MT-GRPO (b)	64	140,544	41.4917	365.21
GradAlign-cosine (c)	16	9,984	37.1324	79.04
GradAlign-cosine (c)	24	14,976	38.5165	119.16
GradAlign-cosine (c)	32	19,968	39.0999	≈159.87
GradAlign-cosine (c)	40	24,960	37.9296	199.55
GradAlign-cosine (c)	64	39,936	36.5972	323.45

Scope (a) covers completed update segments of GRPO, CoGR, and CoGR-Mean and excludes startup, evaluation, and data preparation. Scope (b) is the measured four-GPU checkpoint cost of MT-GRPO. Scope (c) multiplies GradAlign-cosine's four reserved training GPUs by the wall time from run start to checkpoint, so it includes setup and scoring. Because the scopes differ, GPU-hours are not directly comparable across methods. Under scope (a), CoGR uses about 1.75 times the GPU-hours of GRPO. In the first 32 updates of the shared seed, gradient capture takes 1,613.92 seconds of a 3,766.65-second update on average, or 42.85%.

B.10 Development Sweep and Longer-Horizon Continuation

Table B11 reports paired Mean effects from an earlier two-environment development sweep with training seeds `2026202601` and `2026202602`. Each entry compares CoGR with GRPO trained

at the same precision and learning rate. Table B12 reports the FP32, $\beta = 0.25$, learning-rate 2×10^{-5} configuration at 64 and 128 updates.

Table B11: **Development sweep in the Search–TextCraft setting.** Two development seeds and four attempts per problem. Entries are CoGR minus GRPO in pp, and the last three columns average the two seeds.

Precision	β	Policy LR	Updates	Seed 1	Seed 2	Search	TextCraft	Mean
BF16	0.50	10^{-5}	32	−0.8609	−0.9498	−0.7813	−1.0294	−0.9053
BF16	0.50	10^{-5}	64	−0.2941	+0.2604	+0.2604	−0.2941	−0.0169
FP32	0.50	10^{-5}	32	+1.7433	−0.2145	+1.8229	−0.2941	+0.7644
FP32	0.50	10^{-5}	64	+0.5668	+1.1428	+1.5625	+0.1471	+0.8548
FP32	0.25	10^{-5}	32	+1.1091	+1.3358	+1.5625	+0.8824	+1.2224
FP32	0.25	10^{-5}	64	+1.2347	−0.5208	+1.3021	−0.5882	+0.3569
FP32	0.25	2×10^{-5}	32	−0.5545	−3.2935	−2.0833	−1.7647	−1.9240

Longer-horizon continuation. Both methods continue from their own complete 64-update states, including the optimizer, scheduler, random-number, and sampler states, to 128 updates. All entries in Table B12 use 16 attempts per problem with the two-environment 16-attempt seed schedule of Appendix B.4. From 64 to 128 updates, the Mean of CoGR rises from 43.21% to 46.87%, and its average paired gain over GRPO grows from +0.44 to +0.89 pp. Both seeds favor CoGR at 128 updates, by +0.95 and +0.83 pp. These runs extend the two development seeds of the sweep and do not add independent training seeds.

Table B12: **Longer-horizon continuation in the Search–TextCraft setting.** FP32 branches, $\beta = 0.25$, learning rate 2×10^{-5}, and 16 attempts per problem. Seeds 1 and 2 are the training seeds of Table B11.

Seed	Updates	Method	Search	TextCraft	Mean
1	64	GRPO	43.4896	37.5735	40.5316
1	64	CoGR	43.4896	36.9118	40.2007
1	128	GRPO	44.0104	40.8824	42.4464
1	128	CoGR	46.8750	39.9265	43.4007
2	64	GRPO	45.1823	44.8529	45.0176
2	64	CoGR	45.3125	47.1324	46.2224
2	128	GRPO	50.2604	48.7500	49.5052
2	128	CoGR	48.1771	52.5000	50.3385