PROCORE: PROTECTING TASK-SPECIFIC GRADIENT CORES FOR MULTI-TASK RLVR

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-task RLVR enables a single model to jointly learn multiple tasks. We find that each task's gradient is highly concentrated, with a small subset of components accounting for most of the gradient norm. We term them gradient cores. Existing methods dynamically adjust task sampling ratios or weights using reward signals or advantage estimates to balance learning progress across tasks. However, they lack explicit protection for gradient cores during gradient aggregation and struggle to characterize each task's remaining learning potential. Accordingly, we propose **ProCore**, a plug-and-play method for multi-task RLVR. ProCore identifies gradient cores by magnitude within each Transformer block and partitions them into exclusive and shared cores based on dimension overlap across tasks. It computes a Remaining Potential Score (RPS) from the gap between current pass@1 and an estimated performance ceiling derived from pass@k. Building on the original mixed gradient, ProCore directly enhances exclusive cores and dynamically weights shared cores using RPS, giving greater protection to tasks with more remaining potential. We evaluate ProCore on eight heterogeneous reasoning tasks and nine benchmarks spanning mathematics, coding, and scientific QA. ProCore achieves higher average scores than all baselines across both evaluation settings and further improves DAPO and MT-GRPO when combined with them.

1 INTRODUCTION

Large language models have recently made significant progress in mathematical reasoning (Shao et al., 2024; Yu et al., 2025), code generation (Zeng et al., 2025), and question answering (Jin et al., 2025; Song et al., 2025), with reinforcement learning serving as a key driver of these advances (Schulman et al., 2017; Shao et al., 2024). However, existing research has primarily focused on single-task or single-domain training (Liu et al., 2025; Zeng et al., 2025), whereas real-world applications often require a single model to handle multiple tasks with diverse solution spaces and varying learning difficulties (Team, 2025). Optimizing for one task may degrade the model's performance on others, making it difficult to achieve improvements across tasks simultaneously (Cai et al., 2026; Luo et al., 2026). Therefore, enabling a single model to improve across these tasks remains a key challenge in developing general-purpose reasoning models (Cheng et al., 2025).

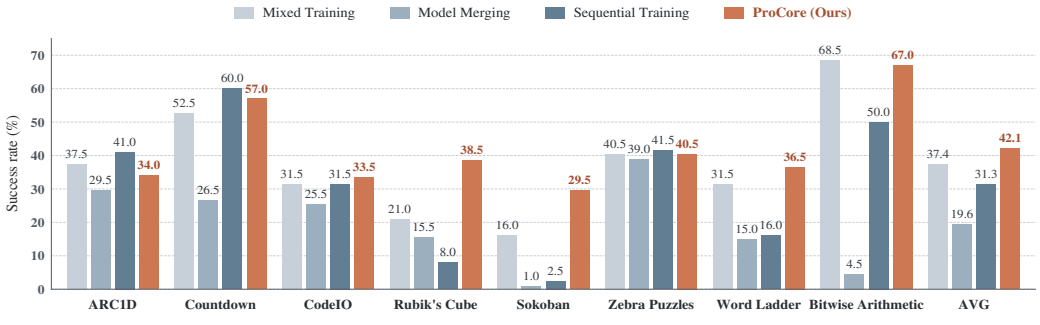


Figure 1: Comparison of three multi-task learning strategies and ProCore on Qwen3-4B.

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.

Three representative paradigms have been explored to address this challenge: **(1) Model Merging** combines models trained independently on individual tasks to consolidate their capabilities into a unified model (Ilharco et al., 2023; Wang et al., 2026). However, parameter updates from different tasks may conflict, degrading the performance of the merged model (Wu et al., 2026a). **(2) Sequential Training** applies reinforcement learning to different tasks in succession (Chen et al., 2025b; Lai et al., 2025; Luo et al., 2026). Nevertheless, this paradigm is sensitive to task ordering, and training in subsequent tasks can compromise previously acquired capabilities, potentially leading to catastrophic forgetting (Cai et al., 2026). **(3) Mixed Training** jointly optimizes a single model on a mixture of data from multiple tasks. It is a widely adopted paradigm for training general-purpose reasoning models (Team, 2025; Team et al., 2025; GLM, 2026) and forms the basis of our approach. Figure 1 compares these three paradigms with our proposed method on eight tasks.

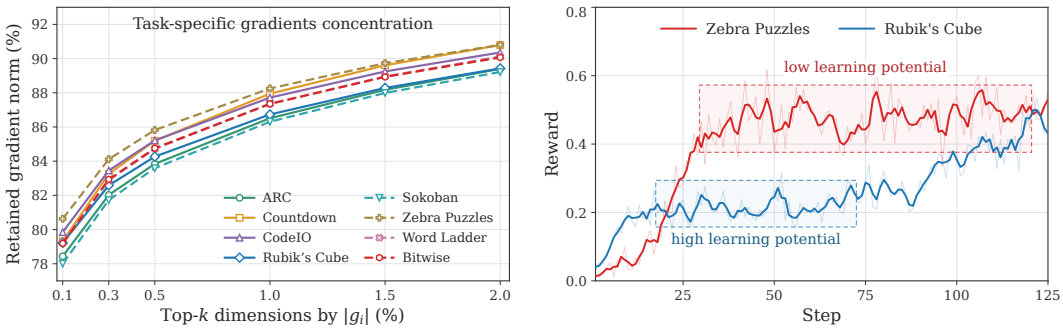
Recent studies dynamically adjust task weights or sampling ratios based on changes in task rewards or advantage statistics to mitigate differences in learning progress across tasks during mixed training (Chen et al., 2025a; Wang et al., 2025; Liu et al., 2026; Yu et al., 2026; Ramesh et al., 2026). These methods essentially adjust the contribution of each task's gradient to the mixed gradient. However, they have two limitations: **(1) Lack of explicit protection for task-specific gradient cores.** Our analysis reveals that, in multi-task RLVR, each task's gradient is highly unevenly distributed across dimensions. As shown in Figure 2(a), a small subset of high-magnitude gradient components accounts for most of each task's gradient norm. We refer to these components as gradient cores. These cores are substantially attenuated in the mixed gradient, yet existing methods do not explicitly protect them. **(2) Insufficient characterization of each task's remaining learning potential.** As illustrated in Figure 2(b), slow improvement on a task may indicate either convergence or insufficient optimization despite remaining room for improvement. However, existing methods rely solely on recent learning progress or advantage statistics to determine whether a task requires further training, making it difficult to distinguish accurately between these two cases.

To address these two limitations, we propose **ProCore**, a plug-and-play method for multi-task RLVR that combines **gradient cores identification and partitioning** with **remaining learning potential estimation**. Specifically, ProCore first identifies each task's gradient cores within each Transformer block based on gradient magnitude, and partitions them into exclusive and shared cores according to whether their corresponding dimensions overlap across tasks. Exclusive cores are selected by only one task, whereas shared cores are selected by multiple tasks. ProCore then constructs an estimated performance ceiling for each task from recent pass@k and computes a Remaining Potential Score (RPS) from the gap between current pass@1 and this estimate to assess the task's remaining learning potential. ProCore directly enhances exclusive cores, while using RPS to dynamically weight shared cores, giving larger weights to tasks with greater remaining potential. Finally, these enhancement terms are added to the original Mixed Training gradient, thereby explicitly protecting each task's gradient cores. Compared with multi-task optimization methods such as PCGrad (Yu et al., 2020) and CAGrad (Liu et al., 2021) that operate over the full gradient space, **ProCore performs only lightweight element-wise operations on the selected gradient cores, substantially reducing memory and computational overhead.**

We evaluate ProCore on eight heterogeneous reasoning tasks and nine benchmarks spanning mathematics, coding, and scientific question answering. ProCore improves the average score over Mixed Training by 4.2 and 3.4 percentage points on Qwen3-4B and Qwen3-8B, respectively. It also achieves the highest average score across the nine benchmarks in the three-domain evaluation. Furthermore, integrating ProCore with DAPO (Yu et al., 2025) and MT-GRPO (Ramesh et al., 2026) yields further performance gains, highlighting its compatibility with existing RLVR methods.

In summary, our contributions are as follows:

- We reveal that task-specific gradients in multi-task RLVR are highly concentrated: the top 1% of components by magnitude retain **87.3%** of each task's gradient norm on average across eight tasks. This observation motivates explicit protection of gradient cores.
- We propose ProCore, a plug-and-play method that identifies and partitions gradient cores and protects them using remaining learning potential estimated by RPS.
- We evaluate ProCore on eight heterogeneous reasoning tasks and nine benchmarks across three domains. Compared with Mixed Training on Qwen3-4B, ProCore improves the average score by **4.2** percentage points and the score on Rubik's Cube by **19.7** percentage points. It further improves DAPO and MT-GRPO.



(a) Task-specific gradients concentration.

(b) Learning progress and remaining potential.

Figure 2: Gradient concentration and learning progress in multi-task RLVR on Qwen3-4B.

2 PRELIMINARIES

2.1 MULTI-TASK RLVR

We consider a set of N reasoning tasks, denoted by $\mathcal{T} = \{1, \ldots, N\}$. Each task i is associated with a prompt distribution $\mathcal{D}_i$ and a verifiable reward function R_i. Given a prompt $q \sim \mathcal{D}_i$, the policy π_θ generates a response $y \sim \pi_\theta(\cdot \mid q)$ and receives a reward $R_i(q, y)$. The expected reward for task i is defined as

$$J_i(\theta) = \mathbb{E}_{q \sim \mathcal{D}_i,\, y \sim \pi_\theta(\cdot|q)} \left[R_i(q, y) \right]. \tag{1}$$

Given nonnegative task weights α_i that sum to one, multi-task RLVR aims to train a shared policy by maximizing the weighted expected reward:

$$\max_\theta J_{\mathrm{MT}}(\theta) = \sum_{i=1}^{N} \alpha_i J_i(\theta). \tag{2}$$

2.2 POLICY GRADIENT OPTIMIZATION

Policy gradient methods, such as REINFORCE (Williams, 1992) and PPO (Schulman et al., 2017), use reward feedback from sampled responses to update policy parameters and maximize expected reward. By the policy gradient theorem (Sutton et al., 1999), the gradient of the expected reward for task i can be expressed as

$$\nabla_\theta J_i(\theta) = \mathbb{E}_{q \sim \mathcal{D}_i,\, y \sim \pi_\theta(\cdot|q)} \left[\nabla_\theta \log \pi_\theta(y \mid q)\, A_i(q, y) \right], \tag{3}$$

where $A_i(q, y)$ denotes the advantage of response y relative to a baseline.

In this work, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to optimize the policy. Let $\mathcal{L}_i(\theta)$ denote the policy surrogate loss for task i. For given task weights, the joint loss under mixed training is

$$\mathcal{L}_{\mathrm{MT}}(\theta) = \sum_{i=1}^{N} \alpha_i \mathcal{L}_i(\theta). \tag{4}$$

Defining the task-specific gradient as $g_i \triangleq \nabla_\theta \mathcal{L}_i(\theta)$, the corresponding base gradient is

$$g_{\mathrm{base}} \triangleq \nabla_\theta \mathcal{L}_{\mathrm{MT}}(\theta) = \sum_{i=1}^{N} \alpha_i g_i. \tag{5}$$

3 PROCORE

In this section, we introduce ProCore, a plug-and-play method for multi-task RLVR. To mitigate the attenuation of each task's primary update signals when computing the mixed gradient, ProCore first identifies and enhances the gradient cores that carry these signals (Section 3.2). To account for differences in learning progress across tasks, ProCore assigns protection weights based on each task's remaining learning potential when gradient cores overlap, giving higher weights to tasks with

more room for improvement (Section 3.3). ProCore thus combines direct enhancement of exclusive cores with weighted enhancement of shared cores, adding both to the original mixed gradient (Section 3.4). Figure 3 illustrates the overall framework of ProCore.

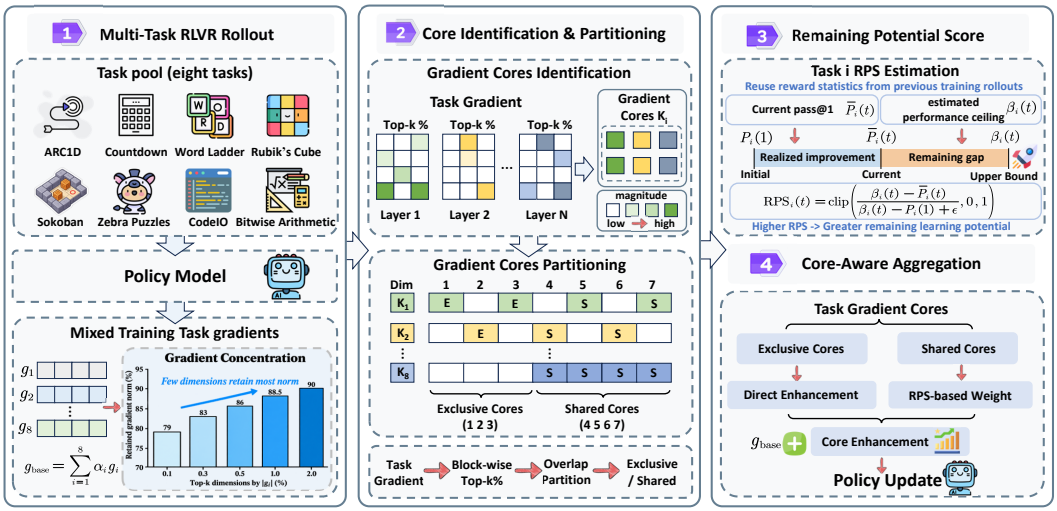


Figure 3: Overview of the ProCore framework.

3.1 The Analysis of Multi-Task RLVR

Task-specific gradient concentration. Task gradients in multi-task RLVR are aggregated through weighted averaging, where each task's gradient is scaled by its corresponding weight (i.e., $1/N$ for an equally weighted mixture of N tasks). Meanwhile, gradient components from different tasks are summed along the same dimensions, and opposing contributions can be further attenuated through cancellation. If each task's primary update signal is concentrated in a small subset of gradient components, with limited overlap in the corresponding dimensions across tasks, selectively protecting these components during aggregation can help preserve that signal. Prior studies have shown that parameter updates in single-task RLVR are highly concentrated in a small subset of dimensions, while most parameters remain nearly unchanged (Zhu et al., 2025; Wu et al., 2026b). Our analysis of task-specific gradient distributions in multi-task RLVR reveals a similar pattern. As shown in Figure 2(a), a small subset of high-magnitude gradient components accounts for most of each task's gradient norm. We refer to these components as the task's gradient cores, which we identify and protect when computing the mixed gradient(Section 3.2).

Learning progress and remaining potential. Learning progress often varies substantially across tasks in multi-task RLVR. We seek to give greater protection to tasks with more remaining learning potential. However, existing methods based on recent progress or advantage statistics struggle to determine whether slow progress reflects convergence or insufficient optimization. We therefore need to assess each task's remaining learning potential. Prior studies have shown that RLVR typically improves pass@1 substantially while producing relatively small changes in pass@k, gradually narrowing the gap between them during training (Chen et al., 2025c; Peng et al., 2025; Yu, 2025). Motivated by this observation, we construct an estimated performance ceiling for each task using its recent pass@k. We then define the Remaining Potential Score (RPS) based on the gap between current pass@1 and this estimate to quantify each task's remaining learning potential. The formal definition is provided in Section 3.3. Figure 4 shows that RPS decreases at markedly different rates across tasks, further demonstrating substantial differences in their learning progress.

3.2 Gradient Cores Identification and Partitioning

ProCore identifies each task's gradient cores and partitions them into exclusive and shared cores based on dimension overlap across tasks. Let $g_{i,\ell}$ denote the gradient of task i over all parameters in the ℓ-th Transformer block. For a model with L blocks, task i's gradient cores are indexed by

$$K_i = \bigcup_{\ell=1}^{L} \mathrm{Top}_\kappa\big(|g_{i,\ell}|\big), \tag{6}$$

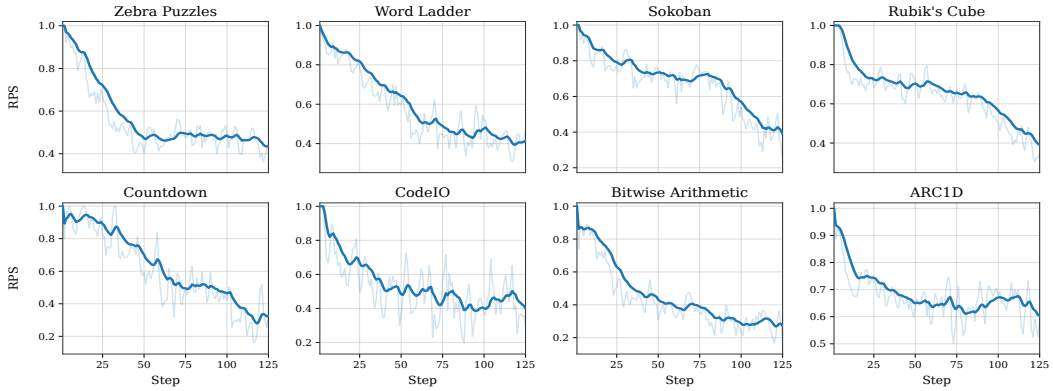


Figure 4: Changes in remaining potential score across eight tasks on Qwen3-4B.

where Top_κ returns the dimension indices of the top $\kappa\%$ of gradient components by magnitude within each block. This block-wise selection reduces the effect of gradient scale differences across blocks and avoids excessive concentration of cores in a few blocks. Comparisons with alternative selection strategies and further analysis are provided in Appendix A.1.

To partition the cores, we count how many tasks select each dimension p:

$$m(p) = \sum_{j \in \mathcal{T}} \mathbf{1}[p \in K_j]. \tag{7}$$

For $p \in K_i$, the component $g_i(p)$ is an exclusive core of task i if $m(p) = 1$ and a shared core if $m(p) > 1$. Figure 5 shows the core composition across eight tasks at $\kappa = 1.1\%$ on Qwen3-4B. Exclusive cores form the largest category for every task, while sharing occurs mainly among small groups of tasks.

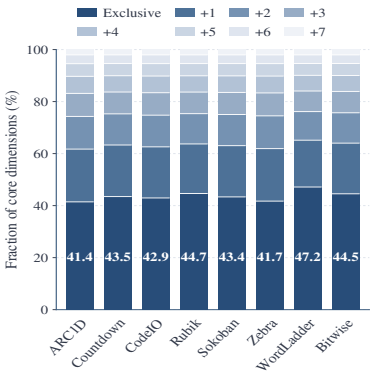


Figure 5: Composition of gradient cores across eight tasks at $\kappa = 1.1\%$.

3.3 Remaining Learning Potential Estimation

ProCore constructs an estimated performance ceiling for each task from recent pass@k and updates it dynamically throughout training. It uses the gap between current pass@1 and this estimate to assess each task's remaining learning potential. At training step t, let $\bar{P}_i(t)$ denote the recent pass@1 for task i. We use $\beta_i(t)$ to denote the estimated performance ceiling and $P_i(1)$ to denote pass@1 at the start of training. We define the Remaining Potential Score (RPS) for task i as

$$\mathrm{RPS}_i(t) = \mathrm{clip}\left(\frac{\beta_i(t) - \bar{P}_i(t)}{\beta_i(t) - P_i(1) + \epsilon}, 0, 1\right), \tag{8}$$

where $\epsilon > 0$ is a numerical stability term. The score measures the fraction of potential improvement that remains unrealized relative to the gap between initial performance and the current estimated performance ceiling. A higher RPS indicates greater remaining learning potential. Details of how $\bar{P}_i(t)$ and $\beta_i(t)$ are estimated are provided in Appendix B.1.

3.4 Core-Aware Gradient Aggregation

ProCore uses each task's remaining learning potential to guide the enhancement of its gradient cores on top of the original mixed gradient. At training step t, the set of tasks selecting a core component at dimension p is

$$S_p = \{i \in \mathcal{T} \mid p \in K_i\}, \tag{9}$$

where $\mathcal{T}$ is the task set. For $S_p \neq \varnothing$, ProCore applies a softmax to these tasks' RPS values to obtain their relative protection weights:

$$\omega_i(p) = \frac{\exp(\mathrm{RPS}_i(t)/\tau)}{\sum_{j \in S_p} \exp(\mathrm{RPS}_j(t)/\tau)}, \quad i \in S_p, \tag{10}$$

where $\tau > 0$ is a temperature parameter set to 0.8 by default. The weighted core gradients are then added to the original mixed gradient:

$$g_{\text{final}}(p) = \begin{cases} g_{\text{base}}(p), & S_p = \varnothing, \\ g_{\text{base}}(p) + \rho \sum_{i \in S_p} \omega_i(p) g_i(p), & S_p \neq \varnothing. \end{cases} \tag{11}$$

Here, $g_{\text{base}}(p)$ and $g_i(p)$ are the components of the original mixed gradient and task i's gradient at dimension p, respectively, and $\rho \geq 0$ controls the enhancement strength.

When $S_p = \{i\}$, $\omega_i(p) = 1$, so task i's exclusive core is enhanced directly. When $|S_p| > 1$, tasks with higher RPS receive larger weights, preferentially preserving their learning signals on shared cores. The original mixed gradient remains unchanged along non-core dimensions.

4 Experiment

4.1 Experimental Settings

Tasks and Datasets. We evaluate ProCore on eight heterogeneous, automatically verifiable tasks from Reasoning Gym (Stojanovski et al., 2025): ARC1D, Countdown, CodeIO, Rubik's Cube (Rubik), Sokoban, Zebra Puzzles (Zebra), Word Ladder (WordLadder), and Bitwise Arithmetic (Bitwise). Each task contains 10,000 training examples and 200 evaluation examples. To ensure a fair comparison, all methods use the same task configurations and data splits. Task descriptions and detailed difficulty settings are provided in Appendix C.1.

Baselines. We compare ProCore with several representative baselines. Mixed Training samples equally from all tasks and jointly trains on mixed multi-task data; Sequential Training trains on different tasks in succession; Model Merging merges models trained independently on individual tasks; and MOPD (Ma et al., 2026) first trains a separate teacher model for each task and then consolidates their capabilities into a single model through multi-teacher online distillation. We also compare methods that dynamically adjust task sampling ratios or weights. For dynamic sampling, we adopt SEC (Chen et al., 2025a) and DADS (Liu et al., 2026), and introduce DynaSample, which adjusts task sampling ratios based on each task's RPS. For dynamic task weighting, we adopt FAMO (Liu et al., 2023a), which adjusts task weights according to their optimization progress. We also include two gradient-based multi-task optimization methods, PCGrad (Yu et al., 2020) and CAGrad (Liu et al., 2021). Further details of the baselines are provided in the Appendix C.2.

Training Details. We train Qwen3-4B and Qwen3-8B with GRPO using the verl (Sheng et al., 2025) and compute rewards using task-specific rule-based verifiers. For ProCore, we select the top 1.1% of gradient components by magnitude within each Transformer block as gradient cores and set $\tau = 0.8$ and $\rho = 0.1$. Further training details are provided in Appendix C.3. The main results in Table 1 are averaged over three random seeds, while all other experiments are conducted with a single random seed.

4.2 Main Results

Table 1 presents the main results comparing ProCore with the baselines across eight reasoning tasks. We make two observations: **(1) ProCore achieves better average performance across model scales.** ProCore improves the average score over Mixed Training by 4.2 and 3.4 percentage points on Qwen3-4B and Qwen3-8B, respectively. It also outperforms the strongest baseline MOPD by 1.1 percentage points on both models. These results demonstrate its effectiveness across model scales. **(2) ProCore improves performance on multiple tasks.** ProCore achieves the highest scores on Rubik and Sokoban with Qwen3-4B, outperforming Mixed Training by 19.7 and 12.2 percentage points, respectively. It also maintains comparable or better performance on Countdown, Zebra, Bitwise, CodeIO, and WordLadder. Figure 9 further presents the training reward curves of ProCore and Mixed Training for each task on Qwen3-8B. ProCore's reward curves are generally higher than those of Mixed Training on most tasks, indicating that ProCore benefits multiple tasks.

4.3 Generalization of ProCore

To further assess the generalizability of ProCore, we extend our evaluation to mathematical reasoning, code generation, and scientific QA. We construct a multi-domain training dataset from

Table 1: Performance comparison of ProCore and baselines across eight tasks. The abbreviation of "STL" denotes Single-Task Learning. Results are averaged over three random seeds. Bold and underlined values indicate the best and second-best results, respectively.

Methods	ARC1D	Countdown	CodeIO	Rubik	Sokoban	Zebra	WordLadder	Bitwise	*AVG*
Qwen3-4B									
Base Model	11.5	28.5	18.5	4.5	4.0	0.5	9.5	11.0	11.0
STL (ARC1D)	49.2$_{\pm 2.0}$	17.5$_{\pm 2.8}$	21.2$_{\pm 1.5}$	9.3$_{\pm 2.1}$	2.8$_{\pm 0.3}$	3.8$_{\pm 1.3}$	7.2$_{\pm 1.3}$	10.8$_{\pm 0.3}$	15.2$_{\pm 0.4}$
STL (Countdown)	23.2$_{\pm 1.3}$	64.8$_{\pm 0.6}$	19.2$_{\pm 2.4}$	5.3$_{\pm 2.1}$	4.0$_{\pm 0.9}$	2.8$_{\pm 0.6}$	10.7$_{\pm 0.3}$	9.8$_{\pm 1.8}$	17.5$_{\pm 0.7}$
STL (CodeIO)	23.5$_{\pm 2.8}$	24.8$_{\pm 1.3}$	36.0$_{\pm 0.5}$	9.2$_{\pm 3.2}$	3.2$_{\pm 2.6}$	5.7$_{\pm 1.6}$	10.3$_{\pm 2.1}$	10.5$_{\pm 1.3}$	15.4$_{\pm 1.0}$
STL (Rubik)	21.7$_{\pm 0.8}$	22.2$_{\pm 0.8}$	19.2$_{\pm 2.3}$	37.3$_{\pm 0.6}$	2.0$_{\pm 1.3}$	6.8$_{\pm 0.8}$	6.2$_{\pm 2.8}$	9.3$_{\pm 1.0}$	15.6$_{\pm 0.9}$
STL (Sokoban)	16.3$_{\pm 0.8}$	30.5$_{\pm 4.0}$	21.3$_{\pm 0.3}$	4.5$_{\pm 3.3}$	16.8$_{\pm 1.0}$	1.5$_{\pm 0.0}$	16.3$_{\pm 1.8}$	11.5$_{\pm 0.5}$	14.9$_{\pm 0.7}$
STL (Zebra)	23.3$_{\pm 1.8}$	26.7$_{\pm 2.0}$	16.8$_{\pm 1.5}$	9.3$_{\pm 1.2}$	3.7$_{\pm 0.8}$	47.0$_{\pm 0.5}$	8.0$_{\pm 0.5}$	5.8$_{\pm 1.3}$	17.6$_{\pm 0.6}$
STL (WordLadder)	20.3$_{\pm 3.2}$	29.0$_{\pm 0.0}$	21.7$_{\pm 0.6}$	7.2$_{\pm 2.3}$	5.0$_{\pm 0.9}$	1.2$_{\pm 0.6}$	40.0$_{\pm 0.0}$	8.8$_{\pm 0.3}$	16.6$_{\pm 0.2}$
STL (Bitwise)	20.5$_{\pm 2.3}$	31.7$_{\pm 3.3}$	24.3$_{\pm 3.3}$	6.0$_{\pm 1.3}$	3.8$_{\pm 1.0}$	1.0$_{\pm 0.5}$	13.5$_{\pm 1.3}$	68.5$_{\pm 0.5}$	21.2$_{\pm 0.8}$
STL (Per Task)	49.2$_{\pm 2.0}$	64.8$_{\pm 0.6}$	36.0$_{\pm 0.5}$	37.3$_{\pm 0.6}$	16.8$_{\pm 1.0}$	47.0$_{\pm 0.5}$	40.0$_{\pm 0.0}$	68.5$_{\pm 0.5}$	45.0$_{\pm 0.3}$
Mixed Training	37.0$_{\pm 0.7}$	55.2$_{\pm 2.1}$	30.5$_{\pm 0.7}$	19.8$_{\pm 1.0}$	16.0$_{\pm 0.8}$	42.7$_{\pm 3.8}$	33.7$_{\pm 1.6}$	66.7$_{\pm 2.6}$	37.7$_{\pm 0.9}$
Sequential Training	37.5$_{\pm 4.1}$	59.2$_{\pm 1.4}$	31.7$_{\pm 0.8}$	8.2$_{\pm 0.3}$	2.7$_{\pm 0.3}$	**43.0$_{\pm 2.2}$**	17.7$_{\pm 1.8}$	50.5$_{\pm 0.9}$	31.3$_{\pm 0.0}$
Model Merging (Avg)	22.7$_{\pm 1.6}$	32.2$_{\pm 1.4}$	21.3$_{\pm 0.6}$	3.7$_{\pm 1.6}$	4.2$_{\pm 1.2}$	3.8$_{\pm 0.3}$	12.8$_{\pm 1.0}$	11.8$_{\pm 0.6}$	14.1$_{\pm 0.6}$
Model Merging (Task)	30.3$_{\pm 1.4}$	24.7$_{\pm 1.6}$	25.3$_{\pm 0.8}$	17.7$_{\pm 2.6}$	1.2$_{\pm 0.3}$	37.8$_{\pm 1.3}$	15.0$_{\pm 1.0}$	4.2$_{\pm 2.0}$	19.5$_{\pm 0.6}$
PCGrad	38.5$_{\pm 2.6}$	51.5$_{\pm 1.8}$	32.0$_{\pm 0.9}$	31.2$_{\pm 1.5}$	20.3$_{\pm 2.5}$	39.7$_{\pm 1.9}$	36.7$_{\pm 0.3}$	56.5$_{\pm 0.9}$	38.3$_{\pm 0.2}$
CAGrad	38.0$_{\pm 1.7}$	57.2$_{\pm 1.4}$	**34.2$_{\pm 1.2}$**	24.3$_{\pm 2.8}$	18.8$_{\pm 3.3}$	41.3$_{\pm 0.6}$	34.7$_{\pm 1.0}$	**74.0$_{\pm 1.3}$**	40.3$_{\pm 0.6}$
SEC	39.0$_{\pm 5.0}$	52.3$_{\pm 1.1}$	27.5$_{\pm 0.0}$	28.3$_{\pm 4.6}$	6.5$_{\pm 0.7}$	42.5$_{\pm 0.7}$	35.0$_{\pm 0.0}$	49.5$_{\pm 0.7}$	35.1$_{\pm 1.2}$
FAMO	38.2$_{\pm 2.8}$	53.5$_{\pm 1.1}$	33.3$_{\pm 0.2}$	25.8$_{\pm 0.8}$	19.5$_{\pm 0.4}$	41.5$_{\pm 2.3}$	35.5$_{\pm 1.5}$	63.2$_{\pm 0.8}$	38.8$_{\pm 0.6}$
DADS	42.8$_{\pm 3.0}$	55.2$_{\pm 1.0}$	28.7$_{\pm 0.5}$	33.8$_{\pm 3.7}$	10.8$_{\pm 1.7}$	40.2$_{\pm 2.0}$	35.2$_{\pm 1.0}$	70.0$_{\pm 0.4}$	39.6$_{\pm 0.5}$
DynaSample	34.2$_{\pm 0.3}$	52.8$_{\pm 2.6}$	32.7$_{\pm 0.3}$	30.5$_{\pm 4.3}$	12.0$_{\pm 2.8}$	40.8$_{\pm 2.1}$	36.3$_{\pm 2.5}$	66.5$_{\pm 0.9}$	38.2$_{\pm 1.6}$
MOPD	**49.5$_{\pm 1.0}$**	**62.7$_{\pm 1.5}$**	**34.2$_{\pm 0.3}$**	25.8$_{\pm 3.2}$	16.3$_{\pm 0.6}$	40.8$_{\pm 1.9}$	**39.0$_{\pm 1.5}$**	57.7$_{\pm 1.5}$	40.8$_{\pm 0.4}$
ProCore	33.7$_{\pm 2.5}$	55.3$_{\pm 1.4}$	33.2$_{\pm 2.4}$	**39.5$_{\pm 0.8}$**	**28.2$_{\pm 1.0}$**	42.2$_{\pm 2.4}$	37.2$_{\pm 0.9}$	66.2$_{\pm 0.6}$	**41.9$_{\pm 0.4}$**
Qwen3-8B									
Base Model	22.0	29.0	12.0	4.0	2.5	0.0	15.5	8.5	11.7
STL (ARC1D)	56.7$_{\pm 1.8}$	28.0$_{\pm 1.0}$	12.2$_{\pm 1.3}$	9.5$_{\pm 2.3}$	2.2$_{\pm 0.6}$	1.2$_{\pm 0.3}$	13.8$_{\pm 1.6}$	8.0$_{\pm 0.0}$	16.4$_{\pm 0.4}$
STL (Countdown)	24.8$_{\pm 2.3}$	58.3$_{\pm 1.0}$	17.5$_{\pm 2.0}$	6.5$_{\pm 1.8}$	3.8$_{\pm 0.8}$	0.7$_{\pm 0.3}$	15.3$_{\pm 3.0}$	7.7$_{\pm 0.3}$	16.8$_{\pm 0.2}$
STL (CodeIO)	23.5$_{\pm 1.3}$	29.2$_{\pm 1.3}$	36.5$_{\pm 1.5}$	9.2$_{\pm 2.5}$	1.7$_{\pm 0.3}$	2.5$_{\pm 0.5}$	12.3$_{\pm 2.1}$	8.5$_{\pm 1.0}$	15.4$_{\pm 0.3}$
STL (Rubik)	25.2$_{\pm 2.0}$	21.7$_{\pm 1.5}$	14.7$_{\pm 0.6}$	42.0$_{\pm 0.5}$	0.7$_{\pm 0.8}$	2.5$_{\pm 1.0}$	10.2$_{\pm 0.6}$	8.5$_{\pm 1.7}$	15.7$_{\pm 0.2}$
STL (Sokoban)	22.0$_{\pm 2.0}$	32.5$_{\pm 1.8}$	19.3$_{\pm 2.3}$	6.2$_{\pm 1.6}$	28.0$_{\pm 1.3}$	1.3$_{\pm 0.8}$	11.2$_{\pm 2.0}$	9.0$_{\pm 0.5}$	16.2$_{\pm 0.5}$
STL (Zebra)	28.2$_{\pm 2.9}$	29.7$_{\pm 2.0}$	21.0$_{\pm 2.5}$	10.3$_{\pm 3.2}$	1.5$_{\pm 1.0}$	47.5$_{\pm 0.9}$	9.7$_{\pm 1.6}$	8.0$_{\pm 1.0}$	19.5$_{\pm 0.5}$
STL (WordLadder)	20.8$_{\pm 1.6}$	30.7$_{\pm 1.5}$	19.2$_{\pm 0.3}$	5.5$_{\pm 1.3}$	4.7$_{\pm 0.8}$	0.7$_{\pm 0.6}$	39.5$_{\pm 1.7}$	7.3$_{\pm 0.3}$	16.0$_{\pm 0.2}$
STL (Bitwise)	21.5$_{\pm 1.7}$	38.0$_{\pm 1.5}$	9.8$_{\pm 1.2}$	1.7$_{\pm 1.3}$	3.8$_{\pm 0.3}$	0.0$_{\pm 0.0}$	10.8$_{\pm 0.8}$	71.7$_{\pm 1.5}$	19.7$_{\pm 0.2}$
STL (Per Task)	56.7$_{\pm 1.8}$	58.3$_{\pm 1.0}$	36.5$_{\pm 1.5}$	42.0$_{\pm 0.5}$	28.0$_{\pm 1.3}$	47.5$_{\pm 0.9}$	39.5$_{\pm 1.7}$	71.7$_{\pm 1.5}$	47.5$_{\pm 0.5}$
Mixed Training	48.8$_{\pm 2.5}$	52.0$_{\pm 1.4}$	33.8$_{\pm 0.6}$	24.3$_{\pm 0.6}$	19.3$_{\pm 0.5}$	43.2$_{\pm 2.2}$	34.3$_{\pm 0.5}$	68.7$_{\pm 1.4}$	40.6$_{\pm 0.6}$
Sequential Training	49.5$_{\pm 3.3}$	**57.7$_{\pm 0.6}$**	36.3$_{\pm 0.3}$	19.7$_{\pm 4.1}$	28.0$_{\pm 1.0}$	44.2$_{\pm 2.0}$	38.8$_{\pm 1.8}$	54.3$_{\pm 1.6}$	41.1$_{\pm 0.1}$
Model Merging (Avg)	26.0$_{\pm 1.0}$	34.2$_{\pm 0.8}$	22.0$_{\pm 0.0}$	6.0$_{\pm 3.5}$	3.3$_{\pm 2.0}$	2.0$_{\pm 0.5}$	15.5$_{\pm 1.5}$	10.7$_{\pm 0.6}$	15.0$_{\pm 0.5}$
Model Merging (Task)	51.8$_{\pm 2.3}$	38.7$_{\pm 2.3}$	33.7$_{\pm 0.6}$	24.5$_{\pm 4.3}$	6.3$_{\pm 3.9}$	39.8$_{\pm 0.8}$	21.5$_{\pm 2.8}$	9.5$_{\pm 0.9}$	28.2$_{\pm 0.6}$
PCGrad	47.3$_{\pm 2.1}$	52.7$_{\pm 1.5}$	**36.7$_{\pm 0.6}$**	30.7$_{\pm 2.6}$	17.5$_{\pm 0.9}$	44.8$_{\pm 1.6}$	38.3$_{\pm 1.0}$	65.2$_{\pm 1.3}$	41.6$_{\pm 0.6}$
CAGrad	47.0$_{\pm 1.3}$	54.2$_{\pm 1.3}$	35.5$_{\pm 1.3}$	27.5$_{\pm 4.3}$	19.0$_{\pm 1.8}$	42.0$_{\pm 3.8}$	39.0$_{\pm 1.3}$	**75.2$_{\pm 2.5}$**	42.4$_{\pm 1.6}$
SEC	51.3$_{\pm 4.4}$	54.8$_{\pm 0.3}$	35.5$_{\pm 1.3}$	**43.5$_{\pm 0.5}$**	7.0$_{\pm 0.5}$	**45.3$_{\pm 2.6}$**	35.2$_{\pm 0.3}$	51.8$_{\pm 2.3}$	40.6$_{\pm 0.1}$
FAMO	47.0$_{\pm 1.6}$	55.8$_{\pm 1.2}$	36.3$_{\pm 0.2}$	29.8$_{\pm 0.8}$	19.3$_{\pm 2.0}$	41.5$_{\pm 2.2}$	**42.2$_{\pm 2.2}$**	68.2$_{\pm 0.8}$	42.5$_{\pm 0.8}$
DADS	51.3$_{\pm 1.5}$	51.7$_{\pm 2.3}$	36.3$_{\pm 0.5}$	25.8$_{\pm 1.2}$	22.5$_{\pm 0.7}$	41.5$_{\pm 0.8}$	37.5$_{\pm 2.3}$	69.7$_{\pm 2.1}$	42.0$_{\pm 0.9}$
DynaSample	48.7$_{\pm 0.3}$	55.2$_{\pm 0.3}$	34.2$_{\pm 0.8}$	30.5$_{\pm 0.5}$	18.3$_{\pm 2.9}$	37.5$_{\pm 1.3}$	38.8$_{\pm 2.3}$	65.5$_{\pm 0.5}$	41.1$_{\pm 0.3}$
MOPD	50.0$_{\pm 2.2}$	53.8$_{\pm 0.3}$	33.8$_{\pm 0.3}$	29.7$_{\pm 1.3}$	**28.5$_{\pm 1.8}$**	41.7$_{\pm 1.3}$	39.5$_{\pm 1.3}$	66.5$_{\pm 1.7}$	42.9$_{\pm 0.5}$
ProCore	**52.0$_{\pm 3.6}$**	54.8$_{\pm 1.3}$	35.5$_{\pm 1.3}$	34.7$_{\pm 3.0}$	20.2$_{\pm 0.6}$	43.2$_{\pm 2.4}$	38.3$_{\pm 1.2}$	73.2$_{\pm 2.1}$	**44.0$_{\pm 1.1}$**

Guru (Cheng et al., 2025) and evaluate on nine benchmarks: MATH500 (Lightman et al., 2024), AIME 2026 (Dekoninck et al., 2026), HMMT (Dekoninck et al., 2026), and OlympiadBench (He et al., 2024) for mathematical reasoning; HumanEval+ (Liu et al., 2023b), MBPP+ (Liu et al., 2023b), and LiveCodeBench v6 (Jain et al., 2025) for code generation; and GPQA-Diamond (Rein et al., 2023) and SuperGPQA (Du et al., 2026) for scientific QA. We conduct experiments on Qwen3-1.7B, Qwen3-4B, and Qwen3-4B-Base. Dataset and experimental details are provided in Appendix D.1 and Appendix D.2, respectively. As shown in Table 2, ProCore achieves the highest average score among all baseline methods on all three models.

4.4 Ablations and Analysis

Ablation Studies. We evaluate two variants: w/o MCS replaces magnitude-based core selection with uniform random selection at the same ratio, while w/o DTP replaces dynamic task protection with equal task weights for shared-core aggregation. Table 3 shows that both variants achieve lower average scores under otherwise identical settings, supporting the effectiveness of both components.

ProCore goes beyond amplifying update magnitudes. ProCore adds a core enhancement term to the original mixed gradient, which changes the overall update magnitude. To examine whether simply increasing the update magnitude can yield similar gains, we scale the learning rate of Mixed

Table 2: Performance comparison of ProCore and baselines across nine benchmarks. Bold and underlined values indicate the best and second-best results, respectively.

	Mathematics				Code			Scientific QA		
Method	**MATH-500**	**AIME 2026**	**HMMT**	**OlympiadBench**	**HumanEval+**	**MBPP+**	**LCB v6**	**GPQA-Diamond**	**SuperGPQA**	***AVG***
Qwen3-1.7B										
Base Model	71.2	10.0	12.1	39.3	57.9	47.4	16.6	2.0	8.5	29.4
Mixed Training	**77.2**	6.7	**9.1**	40.5	59.8	46.6	14.9	6.6	10.0	30.1
FAMO	75.8	13.3	6.1	38.4	**61.0**	47.9	16.0	**8.1**	11.0	30.8
DADS	76.6	10.0	**9.1**	40.2	60.4	45.0	15.4	6.1	8.5	30.1
SEC	75.8	10.0	**9.1**	41.0	60.4	47.9	**16.6**	7.6	10.0	30.9
DynaSample	71.4	**16.7**	**9.1**	38.3	60.4	**48.1**	15.4	3.0	7.0	29.9
ProCore	**77.2**	10.0	**9.1**	**41.5**	**61.0**	46.0	15.4	7.6	**13.0**	**31.2**
Qwen3-4B										
Base Model	83.6	13.3	18.2	50.1	77.4	58.5	25.1	37.4	36.5	44.5
Mixed Training	86.0	20.0	18.2	**53.9**	78.7	59.5	21.1	41.4	**38.0**	46.3
FAMO	86.8	26.7	18.2	52.4	78.7	59.0	**25.7**	40.9	30.5	46.5
DADS	**88.4**	**30.0**	18.2	53.4	79.9	59.0	**25.7**	37.9	35.0	47.5
SEC	85.4	26.7	12.1	52.7	79.9	60.1	25.1	41.9	34.0	46.4
DynaSample	86.2	20.0	18.2	53.1	80.5	59.8	**25.7**	**46.0**	34.0	47.0
ProCore	86.0	**30.0**	**21.2**	52.5	**81.1**	**60.6**	25.1	**46.0**	34.0	**48.5**
Qwen3-4B-Base										
Base Model	73.6	10.0	3.0	36.5	69.5	8.7	17.1	15.7	13.0	27.5
Mixed Training	76.6	**13.3**	**9.1**	39.3	76.2	58.2	18.3	14.1	12.0	35.2
FAMO	77.0	6.7	3.0	**41.7**	73.8	**61.1**	**20.6**	18.2	10.0	34.7
DADS	77.0	10.0	6.1	39.8	74.4	52.4	18.9	15.7	11.0	33.9
SEC	78.2	10.0	6.1	38.7	**79.3**	56.6	17.7	17.7	**15.5**	35.5
DynaSample	76.2	6.7	6.1	41.2	74.4	57.9	18.3	**18.7**	11.5	34.6
ProCore	**78.8**	10.0	6.1	41.4	77.4	59.0	19.4	17.2	14.5	**36.0**

Table 3: Ablation results of ProCore. Bold indicates the best results.

Methods	**ARC1D**	**Countdown**	**CodeIO**	**Rubik**	**Sokoban**	**Zebra**	**WordLadder**	**Bitwise**	***AVG***
Qwen3-4B									
ProCore	34.0	**57.0**	33.5	38.5	**29.5**	**40.5**	36.5	67.0	**42.1**
w/o MCS	40.5	**57.0**	**34.0**	**43.5**	14.5	38.5	**37.0**	59.0	40.5
w/o DTP	**44.0**	54.5	29.0	34.0	21.0	36.0	**37.0**	**74.0**	41.2
Qwen3-8B									
ProCore	**53.0**	**56.0**	**36.0**	**35.0**	20.5	**44.0**	**39.0**	72.5	**44.5**
w/o MCS	51.5	55.5	**36.0**	24.0	15.5	39.5	37.0	70.5	41.2
w/o DTP	47.0	53.5	34.5	30.0	**24.0**	**44.0**	37.0	**73.5**	42.9

Training by $(1+\rho)$ and denote this baseline as **MT-LR**. As shown in Table 4, ProCore still outperforms this baseline on most tasks. These results indicate that simply increasing the update magnitude does not yield the same performance gains as ProCore.

Table 4: Performance comparison between ProCore and MT-LR. Bold indicates the best results.

Methods	**ARC1D**	**Countdown**	**CodeIO**	**Rubik**	**Sokoban**	**Zebra**	**WordLadder**	**Bitwise**	***AVG***
Qwen3-4B									
MT-LR	**38.0**	52.0	**33.5**	23.0	20.0	34.0	35.5	65.5	37.7
ProCore	34.0	**57.0**	**33.5**	**38.5**	**29.5**	**40.5**	**36.5**	**67.0**	**42.1**
Qwen3-8B									
MT-LR	49.5	53.0	33.0	25.0	20.0	**46.0**	34.0	70.5	41.4
ProCore	**53.0**	**56.0**	**36.0**	**35.0**	**20.5**	44.0	**39.0**	**72.5**	**44.5**

ProCore improves the retention of gradient cores. To examine whether ProCore mitigates the attenuation of task-specific gradient cores in the mixed gradient, we compare the gradient cores retention ratios of Mixed Training and ProCore. This metric measures the signal strength retained in the mixed gradient along each task's core gradient direction, with its definition provided in Appendix B.2. As shown in Figure 6, ProCore achieves higher gradient cores retention ratios than Mixed Training across all eight tasks on Qwen3-4B, indicating that it more effectively preserves task-specific core update signals in the mixed gradient.

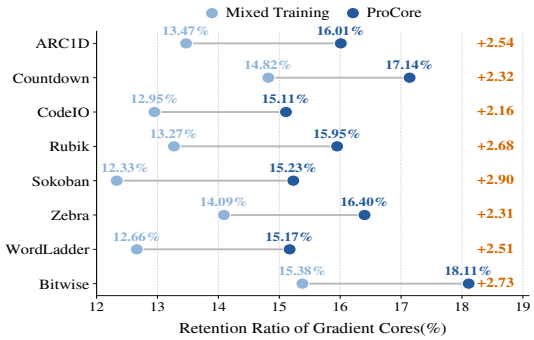


Figure 6: Comparison of gradient cores retention ratios between Mixed Training and ProCore.

ProCore is plug-and-play. We integrate ProCore with DAPO (Yu et al., 2025) and MT-GRPO (Ramesh et al., 2026) on Qwen3-4B and train all four configurations for 75 steps. DAPO and DAPO+ProCore each take approximately the same training time as ProCore in the main experiments, while MT-GRPO and MT-GRPO+ProCore each take approximately 1.2 times as long. As shown in Table 5, incorporating ProCore improves the average scores of DAPO and MT-GRPO by 2.9 and 1.9 percentage points, respectively, with gains on most tasks for both methods. These results demonstrate that ProCore benefits different multi-task RLVR methods.

Table 5: Performance of DAPO and MT-GRPO with and without ProCore on Qwen3-4B. Bold indicates the best results within each method pair.

Methods	ARC1D	Countdown	CodeIO	Rubik	Sokoban	Zebra	WordLadder	Bitwise	*AVG*
DAPO	41.0	58.5	**33.0**	17.0	8.5	**40.5**	32.5	**79.0**	38.8
w/ ProCore	**44.5**	**59.0**	**33.0**	**23.5**	**20.0**	40.0	**37.5**	76.0	**41.7**
MT-GRPO	32.0	**46.5**	**32.0**	32.5	44.5	**35.5**	32.0	56.0	38.9
w/ ProCore	**32.5**	43.5	30.5	**43.0**	**50.0**	35.0	**33.0**	**58.5**	**40.8**

5 Related Work

Gradient-based multi-task optimization. Gradient-based multi-task optimization aims to balance learning across tasks and mitigate inter-task interference by coordinating task updates to shared parameters. GradNorm (Chen et al., 2018) and FAMO (Liu et al., 2023a) balance optimization across tasks through adaptive weighting; MGDA (Sener & Koltun, 2018) seeks a common descent direction through multi-objective optimization; PCGrad (Yu et al., 2020) mitigates conflicts through gradient projection; CAGrad (Liu et al., 2021) accounts for the worst local improvement across tasks when optimizing the average loss; and Nash-MTL (Navon et al., 2022) determines the joint update direction through Nash bargaining. In multi-task reinforcement learning for LLMs, MGS (Cai et al., 2026) addresses gradient conflicts within Transformer modules, while CGPO (Liang et al., 2026) uses cross-domain curvature information to promote implicit gradient alignment. Prior studies have also found that parameter updates in RLVR are concentrated in a small subset of dimensions (Zhu et al., 2025; Wu et al., 2026b). We further observe a similar concentration pattern in task-specific gradients in multi-task RLVR. This observation motivates targeted protection of the gradient components that carry each task's primary update signal.

Multi-task RLVR. Multi-task RLVR jointly optimizes a model over tasks from multiple domains to improve its general reasoning capabilities (Cheng et al., 2025). To accommodate differences in learning progress across tasks, SEC (Chen et al., 2025a), DUMP (Wang et al., 2025), DADS (Liu et al., 2026), and PAC (Yu et al., 2026) dynamically adjust task sampling ratios based on training feedback to optimize the allocation of training resources across tasks. Specifically, SEC and DUMP primarily rely on advantage statistics, DADS uses task rewards, and PAC combines advantage statistics with recent reward gains. MT-GRPO (Ramesh et al., 2026) dynamically updates task weights and uses proportion-preserving sampling to align each task's contribution to the effective gradient with its target weight. These methods regulate each task's contribution to the mixed gradient by adjusting task sampling ratios or task weights, but do not explicitly protect task-specific gradient cores. ProCore distinguishes task-specific gradient cores and further enhances them on top of the original mixed gradient to preserve each task's primary update signal.

6 Conclusion

We propose ProCore, a plug-and-play method for multi-task RLVR that identifies task-specific gradient cores by magnitude within each Transformer block and partitions them into exclusive and shared cores based on cross-task dimension overlap. On top of the original mixed gradient, ProCore directly enhances exclusive cores and weights shared-core enhancements using RPS, which estimates remaining learning potential from the gap between current pass@1 and a performance ceiling derived from recent pass@k. Experiments on eight heterogeneous reasoning tasks and nine benchmarks spanning mathematics, coding, and scientific QA show that ProCore outperforms all baselines in average score across the evaluated model settings and further improves DAPO and MT-GRPO.

AI USE STATEMENT

In this work, we used ChatGPT to assist with manuscript writing and language editing, focusing on grammar, wording, sentence structure, clarity, and coherence. We independently evaluated its suggestions and reviewed and revised all AI-assisted text to ensure that it accurately reflected our research. We also formulated the research problem, designed the method, conducted the experiments, and analyzed the results ourselves. We take full responsibility for the accuracy and integrity of the final manuscript, including its technical claims, experimental results, and conclusions.

ETHICS STATEMENT

This work aims to improve multi-task RLVR for large language models. We conduct experiments using procedurally generated reasoning tasks and existing datasets for mathematics, code generation, and scientific question answering. Our research does not involve human subjects or the collection of sensitive personal information. We describe the dataset construction, task configurations, and evaluation procedures in the paper to support transparency and reproducibility.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide the mathematical formulation of ProCore in the main text and training pseudocode in Algorithm 1, with details of RPS estimation in Appendix B.1. We describe the data sources, dataset construction, and task configurations for the main and generalization experiments in Appendices C.1 and D.1, respectively. We specify the models, training hyperparameters, and experimental settings in Appendices C.3 and D.2. We also describe the baseline implementations and configurations in Appendix C.2 to help readers reproduce our results.

REFERENCES

Min Cai, Yu Liang, Longzheng Wang, Yan Wang, Yueyang Zhang, Long Xia, Zhiyuan Sun, Xi Ye, and Daiting Shi. Advancing general-purpose reasoning models with modular gradient surgery. *CoRR*, abs/2602.02301, 2026. doi: 10.48550/ARXIV.2602.02301. URL https://doi.org/10.48550/arXiv.2602.02301.

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. *CoRR*, abs/2505.14970, 2025a. doi: 10.48550/ARXIV.2505.14970. URL https://doi.org/10.48550/arXiv.2505.14970.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/9fc2aa7f8a66eed231007b8bee4485cb-Abstract-Conference.html.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 793–802. PMLR, 2018. URL http://proceedings.mlr.press/v80/chen18a.html.

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? In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025c. URL http://papers.nips.cc/paper_files/paper/2025/hash/537d5aa768c2d534016a4d06f87bc8fb-Abstract-Conference.html.

Jorge (Zhoujun) Cheng, Shibo Hao, Tianyang Liu, Fan Zhou, Yutao Xie, Feng Yao, Yuexin Bian, Nilabjo Dey, Yonghao Zhuang, Yuheng Zha, Yi Gu, Kun Zhou, Yuqi Wang, Yuan Li, Richard Fan, Jianshu She, Chengqian Gao, Abulhair Saparov, Taylor W. Killian, Haonan Li, Mikhail Yurochkin, Eric P. Xing, Zhengzhong Liu, and Zhiting Hu. Revisiting reinforcement learning for LLM reasoning from A cross-domain perspective. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/c2f71567cd53464161cab3336e8fc865-Abstract-Datasets_and_Benchmarks_Track.html.

Jasper Dekoninck, Nikola Jovanovic, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin T. Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *CoRR*, abs/2605.00674, 2026. doi: 10.48550/ARXIV.2605.00674. URL https://doi.org/10.48550/arXiv.2605.00674.

Xeron Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *Advances in Neural Information Processing Systems*, 38, 2026.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 3828–3850. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.211. URL https://doi.org/10.18653/v1/2024.acl-long.211.

Gabriel Ilharco, Marco Túlio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL https://doi.org/10.48550/arXiv.2503.09516.

Song Lai, Haohan Zhao, Rong Feng, Changyi Ma, Wenzhuo Liu, Hongbo Zhao, Xi Lin, Dong Yi, Min Xie, Qingfu Zhang, Hongbin Liu, Gaofeng Meng, and Fei Zhu. Reinforcement fine-tuning naturally mitigates forgetting in continual post-training. *CoRR*, abs/2507.05386, 2025. doi: 10.48550/ARXIV.2507.05386. URL https://doi.org/10.48550/arXiv.2507.05386.

Xize Liang, Lin Yang, Jie Wang, Rui Liu, Yang Lu, Jinliang Zeng, Hanzhu Chen, Dong Li, and Jianye Hao. Boosting multi-domain reasoning of llms via curvature-guided policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 18878–18890, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/9d27fdf2477ffbff837d73ef7ae23db9-Abstract.html.

Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. FAMO: fast adaptive multitask optimization. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a. URL http://papers.nips.cc/paper_files/paper/2023/hash/b2fe1ee8d936ac08dd26f2ff58986c8f-Abstract-Conference.html.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Yang Liu, Jiaqi Li, and Zilong Zheng. Rulereasoner: Reinforced rule-based reasoning via domain-aware dynamic sampling. In *International Conference on Learning Representations*, volume 2026, pp. 70620–70648, 2026.

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. *CoRR*, abs/2503.20783, 2025. doi: 10.48550/ARXIV.2503.20783. URL https://doi.org/10.48550/arXiv.2503.20783.

Lirui Luo, Guoxi Zhang, Hongming Xu, Rongqing Li, Cong Fang, and Lifeng Fan. Continual reasoning gym: Diagnosing and harnessing shared reasoning in continual RLVR. *CoRR*, abs/2608.18574, 2026. doi: 10.48550/ARXIV.2608.18574. URL https://doi.org/10.48550/arXiv.2608.18574.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, Jinhao Dong, Zhifang Sui, and Fuli Luo. MOPD: multi-teacher on-policy distillation for capability integration in LLM post-training. *CoRR*, abs/2606.30406, 2026. doi: 10.48550/ARXIV.2606.30406. URL https://doi.org/10.48550/arXiv.2606.30406.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16428–16446. PMLR, 2022. URL https://proceedings.mlr.press/v162/navon22a.html.

Ruotian Peng, Yi Ren, Zhouliang Yu, Weiyang Liu, and Yandong Wen. Beyond the sampled token: Preserving candidate support in rlvr. *arXiv preprint arXiv:2510.14807*, 2025.

Shyam Sundhar Ramesh, Xiaotong Ji, Matthieu Zimmer, Sangwoong Yoon, Zhiyong Wang, Haitham Bou-Ammar, Aurélien Lucchi, and Ilija Bogunovic. Multi-task GRPO: reliable LLM reasoning across tasks. *CoRR*, abs/2602.05547, 2026. doi: 10.48550/ARXIV.2602.05547. URL https://doi.org/10.48550/arXiv.2602.05547.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022, 2023. doi: 10.48550/ARXIV.2311.12022. URL https://doi.org/10.48550/arXiv.2311.12022.

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

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 525–536, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/432aca3a1e345e339f35a30c8f65edce-Abstract.html.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

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 *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

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. *CoRR*, abs/2503.05592, 2025. doi: 10.48550/ARXIV.2503.05592. URL https://doi.org/10.48550/arXiv.2503.05592.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/528d56195a2c77c808494c86fa7c77ad-Abstract-Datasets_and_Benchmarks_Track.html.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Prime Intellect Team, Mika Senghaas, Fares Obeid, Sami Jaghouar, William Brown, Jack Min Ong, Daniel Auras, Matej Sirovatka, Jannik Straube, Andrew Baker, Sebastian Müller, Justus Mattern, Manveer Basra, Aiman Ismail, Dominik Scherm, Cooper Miller, Ameen Patel, Simon Kirsten, Mario Sieg, Christian Reetz, Kemal Erdem, Vincent Weisser, and Johannes Hagemann. INTELLECT-3: technical report. *CoRR*, abs/2512.16144, 2025. doi: 10.48550/ARXIV.2512.16144. URL https://doi.org/10.48550/arXiv.2512.16144.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Haoqing Wang, Xiang Long, Ziheng Li, Yilong Xu, Tingguang Li, and Yehui Tang. To mix or to merge: Toward multi-domain reinforcement learning for large language models. *CoRR*, abs/2602.12566, 2026. doi: 10.48550/ARXIV.2602.12566. URL https://doi.org/10.48550/arXiv.2602.12566.

Zhenting Wang, Guofeng Cui, Kun Wan, and Wentian Zhao. DUMP: automated distribution-level curriculum learning for rl-based LLM post-training. *CoRR*, abs/2504.09710, 2025. doi: 10.48550/ARXIV.2504.09710. URL https://doi.org/10.48550/arXiv.2504.09710.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Chenrui Wu, Zexi Li, Jiajun Bu, Jiangchuan Liu, and Haishuai Wang. Sparsity curse: Understanding RLVR model parameter space from model merging. In Won Kim, Jae-Gil Lee, Kyuseok Shim, James Caverlee, David F. Gleich, Xiangliang Zhang, Qi He, Faisal Farooq, Rómer Rosales, Sayan Ranu, Haixun Wang, Yan Liu, Carl Yang, Leman Akoglu, Nitesh V. Chawla, and Vagelis Papalexakis (eds.), *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, KDD 2026, Jeju Island, Korea, August 9-13, 2026*, pp. 5384–5394. ACM, 2026a. doi: 10.1145/3770855.3817679. URL https://doi.org/10.1145/3770855.3817679.

Chenrui Wu, Zexi Li, Jiajun Bu, Jiangchuan Liu, and Haishuai Wang. Sparsity curse: Understanding rlvr model parameter space from model merging. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5384–5394, 2026b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3fe78a8acf5fda99de95303940a2420c-Abstract.html.

Yang Yu. Pass@ k metric for rlvr: A diagnostic tool of exploration, but not an objective. *arXiv preprint arXiv:2511.16231*, 2025.

YuanQiang Yu, Yanzhao Zheng, ZhenTao Zhang, Tianze Xu, Chao Ma, JiHuai Zhu, Jiashun Liu, Xinle Deng, Baohua Dong, Hangcheng Zhu, and Ruohui Huang. PAC: progress-augmented advantage curriculum for multi-task reinforcement learning of llms. *CoRR*, abs/2608.30528, 2026. doi: 10.48550/ARXIV.2608.30528. URL https://doi.org/10.48550/arXiv.2608.30528.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. ACECODER: acing coder RL via automated test-case synthesis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 12023–12040. Association for Computational Linguistics,

2025. doi: 10.18653/V1/2025.ACL-LONG.587. URL https://doi.org/10.18653/v1/2025.acl-long.587.

Hanqing Zhu, Zhenyu Zhang, Hanxian Huang, DiJia Su, Zechun Liu, Jiawei Zhao, Igor Fedorov, Hamed Pirsiavash, Zhizhou Sha, Jinwon Lee, David Z. Pan, Zhangyang Wang, Yuandong Tian, and Kai Sheng Tai. The path not taken: RLVR provably learns off the principals. *CoRR*, abs/2511.08567, 2025. doi: 10.48550/ARXIV.2511.08567. URL https://doi.org/10.48550/arXiv.2511.08567.

A Further Analysis of ProCore

A.1 Effect of gradient cores selection strategies

We compare six gradient cores selection strategies on Qwen3-4B: AB-FFN, AB-Attn, AB-Full, PB-FFN, PB-Attn, and PB-Full. AB denotes global selection across all Transformer blocks, while PB denotes independent selection within each block; FFN, Attn, and Full indicate selection from feed-forward parameters, attention parameters, and all parameters, respectively. As shown in Table 6, PB-Full achieves the highest average score.

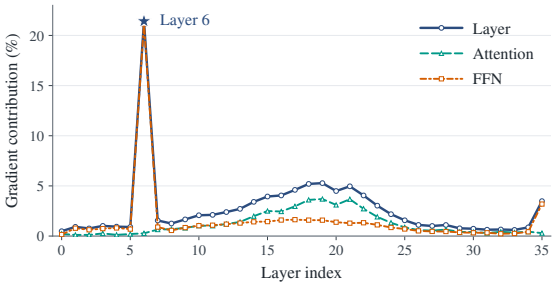


Figure 7: Layer-wise gradient contributions of ProCore on Qwen3-4B.

To analyze the performance differences among selection strategies, we further examine the gradient contributions of individual Transformer blocks and their attention and FFN modules. We measure contribution as the percentage of the full-model squared L_2 gradient norm attributable to each block or module, averaged over tasks and mini-batches. As shown in Figure 7, gradient contributions are highly uneven across blocks, with the block at index 6 accounting for approximately 21% of the total. Global selection may therefore overly concentrate gradient cores in a few blocks with larger gradient scales, whereas block-wise selection helps maintain coverage across blocks. The relative contributions of attention and FFN also vary across blocks: the peak at index 6 is dominated by FFN, while attention contributes more in several middle blocks. Restricting selection to a single module type may thus omit important gradient components in other modules. These observations further support the use of PB-Full.

Table 6: Effect of gradient cores selection strategies on Qwen3-4B. Bold indicates the best results.

Methods	ARC1D	Countdown	CodeIO	Rubik	Sokoban	Zebra	WordLadder	Bitwise	*AVG*
AB-FFN	41.0	60.5	**34.5**	23.5	21.0	41.5	39.5	66.5	41.0
AB-Attn	**44.5**	**63.0**	30.5	29.5	16.0	40.5	38.0	**70.0**	41.5
AB-Full	36.0	55.5	30.5	34.5	17.0	39.5	**42.0**	65.5	40.1
PB-FFN	37.5	55.5	29.0	25.5	21.0	41.0	32.0	66.5	38.5
PB-Attn	38.0	53.0	33.0	37.5	10.5	**43.0**	33.0	68.5	39.6
PB-Full	34.0	57.0	33.5	**38.5**	**29.5**	40.5	36.5	67.0	**42.1**

A.2 The impact of gradient cores ratios κ.

κ controls the gradient cores selection ratio within each Transformer block. A small κ may leave some important gradient components unprotected, whereas an excessively large κ includes more lower-magnitude gradient components for enhancement, potentially increasing interference between tasks. As shown in Table 7, ProCore achieves the highest average score at $\kappa = 1.1\%$, with both smaller and larger ratios yielding lower average performance.

Table 7: Effect of gradient cores ratios κ on Qwen3-4B. Bold indicates the best results.

Methods	ARC1D	Countdown	CodeIO	Rubik	Sokoban	Zebra	WordLadder	Bitwise	*AVG*
ProCore (κ=1.0%)	34.0	54.0	**34.0**	**43.0**	11.5	37.5	36.5	**67.0**	39.7
ProCore (κ=1.1%)	34.0	**57.0**	33.5	38.5	**29.5**	40.5	36.5	**67.0**	**42.1**
ProCore (κ=1.2%)	**37.0**	56.0	30.5	34.5	22.5	**42.5**	**39.5**	63.0	40.7

A.3 The impact of core enhancement coefficient ρ.

ρ controls the strength of gradient cores enhancement. A small ρ provides limited protection for gradient cores, whereas an excessively large ρ may overemphasize the update signals of some tasks, increasing interference between tasks. As shown in Table 8, ProCore achieves the highest average score at $\rho = 0.1$, with both larger and smaller values yielding lower average performance.

Table 8: Effect of core enhancement coefficient ρ on Qwen3-4B. Bold indicates the best results.

Methods	ARC1D	Countdown	CodeIO	Rubik	Sokoban	Zebra	WordLadder	Bitwise	*AVG*
ProCore (ρ=0.05)	**39.0**	54.0	30.5	35.5	15.0	**41.0**	**37.5**	**70.0**	40.3
ProCore (ρ=0.1)	34.0	**57.0**	33.5	38.5	**29.5**	40.5	36.5	67.0	**42.1**
ProCore (ρ=0.2)	38.0	54.0	**35.0**	**43.5**	16.0	**41.0**	33.0	61.5	40.3

A.4 ProCore introduces only negligible additional computation overhead.

Whereas Mixed Training directly computes the mixed gradient, ProCore performs separate backward passes for each task to obtain task-specific gradients, followed by gradient cores identification and weighted enhancement. RPS updates reuse rewards collected during training and require no additional sampling. As shown in Table 9, ProCore incurs only a slight increase in total training time and per-step time compared with Mixed Training, while MOPD requires more than twice the total training time of ProCore. These results show that ProCore adds little training overhead.

Table 9: Training time comparison across methods and model sizes.

Methods	Total (h)	Step (min)
Qwen3-4B		
Mixed Training	14.08	7.04
Model Merging	30.01	15.01
Sequential Training	18.29	9.14
MOPD	36.56	–
ProCore	14.77	7.38
Qwen3-8B		
Mixed Training	12.65	6.33
Model Merging	35.08	17.54
Sequential Training	24.64	12.32
MOPD	41.30	–
ProCore	13.16	6.58

B Supplementary Details of ProCore

B.1 Remaining Learning Potential Estimation

To estimate each task's remaining learning potential, we compare its performance under single and multiple response sampling. Here, pass@1 denotes the success rate of a single response, while pass@k denotes the probability of obtaining at least one correct response among k sampled responses. We use pass@k to construct an estimated performance ceiling and compute the Remaining Potential Score (RPS) from the gap between the current pass@1 and this estimate.

At training step t, let $\mathcal{Q}_i(t)$ denote the set of prompts for task i. For each prompt q, we sample k responses and denote the score of the j-th response by $r_{i,q,j}$. We first compute the mean score across all responses in the batch:

$$A_i(t) = \frac{1}{k|\mathcal{Q}_i(t)|} \sum_{q \in \mathcal{Q}_i(t)} \sum_{j=1}^{k} r_{i,q,j}. \tag{12}$$

We then take the highest score among k responses per prompt and average these maxima:

$$B_i(t) = \frac{1}{|\mathcal{Q}_i(t)|} \sum_{q \in \mathcal{Q}_i(t)} \max_{1 \le j \le k} r_{i,q,j}. \tag{13}$$

When $r_{i,q,j}$ is a binary correctness reward, $A_i(t)$ and $B_i(t)$ correspond to pass@1 and pass@k on the current batch, respectively. We set k=8 to match the GRPO group size, allowing both statistics to be computed directly from responses already sampled during training without additional generation.

To account for sampling fluctuations in individual batches, we use recent training statistics to estimate both current pass@1 and the performance ceiling. Specifically, we use the mean pass@1 over

the three most recent training steps to estimate current pass@1:

$$\bar{P}_i(t) = \frac{1}{\min(3, t-1)} \sum_{h=1}^{\min(3,t-1)} A_i(t-h). \tag{14}$$

We use the maximum pass@k over the last five steps as the estimated performance ceiling:

$$\beta_i(t) = \max_{1 \le h \le \min(5,t-1)} B_i(t-h). \tag{15}$$

Tasks may differ in both their initial performance and the gap between that performance and the estimated performance ceiling. We therefore normalize the remaining gap. Let $P_i(1)$ denote the pass@1 of task i at the start of training. We define RPS as

$$\mathrm{RPS}_i(t) = \mathrm{clip}\left(\frac{\beta_i(t) - \bar{P}_i(t)}{\beta_i(t) - P_i(1) + \epsilon}, 0, 1\right), \tag{16}$$

where $\epsilon > 0$ is a numerical stability term. In implementation, we lower-bound $\beta_i(t)$ by $P_i(1)$ and set $\mathrm{RPS}_i(t) = 0$ whenever $\beta_i(t) - P_i(1) \le \epsilon$. The numerator measures the remaining gap between current performance and the estimated performance ceiling, while the denominator measures the gap between initial performance and the same estimate. A higher RPS indicates greater remaining learning potential. ProCore accordingly assigns higher weights to tasks with higher RPS when aggregating shared gradient cores.

B.2 The Definition of Retention Ratio

To quantify the retention of each task's gradient cores in the mixed gradient, we define the retention ratio using a normalized projection. Let g_i denote the gradient of task i and K_i the set of dimension indices corresponding to its gradient cores. For the aggregated mixed gradient g, the retention ratio for task i is defined as

$$\mathrm{RR}_i(g) = \frac{\sum_{p \in K_i} g(p)\, g_i(p)}{\sum_{p \in K_i} g_i(p)^2 + \epsilon}, \tag{17}$$

where ϵ ensures numerical stability. This metric measures the retained signal strength along the task's core gradient direction, with higher values indicating greater retention.

B.3 Detailed Training Procedure for ProCore

Algorithm 1 summarizes the training procedure. At each step, ProCore computes task-specific GRPO gradients and selects gradient cores by magnitude within each Transformer block. These cores are classified as exclusive or shared according to their coordinate overlap across tasks. The remaining learning potential of each task is estimated using RPS from recent training statistics. Starting from the original mixed gradient, ProCore directly enhances exclusive cores and applies RPS-weighted enhancement to shared cores, while leaving non-core coordinates unchanged. The resulting gradient is then used to update the policy. RPS estimation reuses existing training rollouts and requires no additional response generation.

C Details of Main Experiments

C.1 Tasks and Datasets

We construct datasets for eight tasks from Reasoning Gym (Stojanovski et al., 2025), covering pattern induction, arithmetic reasoning, program execution reasoning, spatial planning, logical reasoning, and word transformations. To match the task difficulty to the capabilities of Qwen3-4B and Qwen3-8B while retaining room for improvement, we adjust the difficulty of each task and provide auxiliary hints for selected tasks. Each task contains 10,000 training examples and 200 evaluation examples, yielding 80,000 training examples and 1,600 evaluation examples in total. All methods at both model scales use the same task configurations and data splits. The task definitions and difficulty settings are detailed below.

Algorithm 1 Training Procedure for ProCore

Require: Policy π_θ; tasks $\mathcal{T}$; weights $\{\alpha_i\}$; initial scores $\{P_i(1)\}$; steps T; $\kappa, \rho, \tau, \epsilon$
Ensure: Updated policy π_θ

Initialize $\mathrm{RPS}_i(1) \leftarrow 1$ for all tasks
for $t = 1, \ldots, T$ **do**
1. Identify gradient cores
  for $i \in \mathcal{T}$ **do**
    Collect rollouts, record reward statistics, and compute $\mathcal{L}_i$
    $g_i \leftarrow \nabla_\theta \mathcal{L}_i$; $K_i \leftarrow \bigcup_{\ell=1}^{L} \mathrm{Top}_\kappa(|g_{i,\ell}|)$
  end for
2. Estimate remaining learning potential
  if $t > 1$ **then**
    Estimate $\bar{P}_i(t)$ and $\beta_i(t)$ from the preceding 3 and 5 available steps, respectively, for all tasks
    $\mathrm{RPS}_i(t) \leftarrow \mathrm{clip}\left(\frac{\beta_i(t) - \bar{P}_i(t)}{\beta_i(t) - P_i(1) + \epsilon}, 0, 1\right), \quad i \in \mathcal{T}$
  end if
3. Enhance and aggregate gradient cores
  $g_{\mathrm{base}} \leftarrow \sum_{i \in \mathcal{T}} \alpha_i g_i$; $g_{\mathrm{final}} \leftarrow g_{\mathrm{base}}$
  for $p \in \bigcup_i K_i$ **do**
    $S_p \leftarrow \{i \in \mathcal{T} \mid p \in K_i\}$
    $\omega_i(p) \leftarrow \frac{\exp(\mathrm{RPS}_i(t)/\tau)}{\sum_{j \in S_p} \exp(\mathrm{RPS}_j(t)/\tau)}, \quad i \in S_p$ ▷ $\omega_i(p) = 1$ *for exclusive cores*
    $g_{\mathrm{final}}(p) \leftarrow g_{\mathrm{base}}(p) + \rho \sum_{i \in S_p} \omega_i(p) g_i(p)$
  end for
  $\theta \leftarrow \textsc{OptimizerStep}(\theta, g_{\mathrm{final}})$
end for
return π_θ

- **ARC1D:** Given input–output examples of one-dimensional grids, the model must infer the underlying transformation rule and apply it to a new input grid. We fix the grid length to 10 cells and provide three input–output demonstration pairs per problem, requiring the model to output the transformed test grid. A representative example is provided in Appendix E.1.
- **Countdown:** Given a set of numbers and a target value, the model must construct an arithmetic expression that evaluates to the target using addition, subtraction, multiplication, and division, with each supplied number used exactly once. We use five numbers per problem, with values ranging from 1 to 100 and integer targets ranging from 1 to 1,000. A representative example is provided in Appendix E.2.
- **CodeIO:** The model predicts a program's output from its task description, input–output specifications, reference code, and a given input, and returns the result as a JSON value. We construct an equal mixture of examples generated from program templates originally labeled with difficulty levels 6 and 7 in the Reasoning Gym CodeIO source data. Each difficulty level contributes 5,000 training examples and 100 evaluation examples. Inputs are generated using the corresponding templates, and reference outputs are obtained by executing the programs. A representative example is provided in Appendix E.3.
- **Rubik's Cube:** Given a textual representation of a scrambled $3 \times 3 \times 3$ Rubik's cube, the model must generate a sequence of moves that restores it to the solved state. We use cubes scrambled with one or two moves, accounting for 70% and 30% of both the training and evaluation sets, respectively. Each prompt specifies the scramble length and requests a solution with the same number of moves. For cubes scrambled with two moves, the prompt additionally identifies the faces involved in a valid solution. A representative example is provided in Appendix E.4.
- **Sokoban:** The model must plan the player's movements through a grid to push a box onto a goal. We use 6×6 boards containing one box and one goal, and select instances with reference solutions of four moves. The prompt explicitly states the required solution length; a response is considered correct only if it consists of four moves and solves the puzzle. A representative example is provided in Appendix E.5.

- **Zebra Puzzles:** The model must combine relational clues to infer the attributes of people living in an ordered sequence of houses. We configure each puzzle with five houses, five people, and five attribute categories, including names. For each generated puzzle, we randomly select a question about an attribute of the person in a specified house and use the corresponding attribute value as the answer. A representative example is provided in Appendix E.6.
- **Word Ladder:** The model must transform a starting word into a target word by changing one letter at a time, with every word in the sequence belonging to the task vocabulary. We use words of four letters and control difficulty through the number of words in the reference transformation chain. Chains containing 4, 5, 6, and 7 words account for 40%, 40%, 10%, and 10% of both the training and evaluation sets, respectively. Chain length includes both the starting and target words. A representative example is provided in Appendix E.7.
- **Bitwise Arithmetic:** The model must evaluate nested expressions over hexadecimal integers involving addition, subtraction, multiplication, and left and right bit shifts. We control difficulty through the number of binary operations in each expression: expressions containing two, three, and four operations account for 45%, 45%, and 10% of both the training and evaluation sets, respectively. Shift amounts range from 0 to 3, and the absolute values of the results require at most 40 bits to represent. The model must return the answer in hexadecimal notation, using an explicit minus sign for negative values. A representative example is provided in Appendix E.8.

C.2 Baselines

We compare ProCore with baselines covering joint training, sequential training, model merging, gradient-based multi-task optimization, dynamic task sampling, adaptive task weighting, and multi-teacher distillation. We implement existing methods following their original papers or official implementations.

- **Mixed Training:** We sample equally from all tasks and jointly train a shared model on the mixed data using GRPO. The task sampling ratios remain fixed throughout training.
- **Sequential Training:** We apply reinforcement learning sequentially in the order Bitwise $\rightarrow$ Zebra $\rightarrow$ Sokoban $\rightarrow$ Rubik $\rightarrow$ WordLadder $\rightarrow$ CodeIO $\rightarrow$ Countdown $\rightarrow$ ARC1D. Each stage uses only the data from the current task and is initialized from the model obtained in the preceding stage.
- **Model Merging(Avg):** Starting from the same initial model, we independently train a model on each task using reinforcement learning, then average the resulting model parameters to obtain a unified model:

$$\theta_{\text{Avg}} = \frac{1}{N}\sum_{i=1}^{N}\theta_i, \tag{18}$$

  where θ_i denotes the parameters of the model trained independently on task i, and $N = 8$ is the number of tasks.
- **Model Merging(Task):** Task Arithmetic defines a task vector as the parameter difference between a task-specific model and their shared initial model, and combines these vectors to integrate capabilities across tasks. We use the same task-specific models as Avg and compute the merged parameters as

$$\theta_{\text{TA}} = \theta_0 + \lambda\sum_{i=1}^{N}(\theta_i - \theta_0), \tag{19}$$

  where θ_0 denotes the parameters of the shared initial model and λ is the merging coefficient. We set $\lambda = 1$.
- **PCGrad:** PCGrad mitigates inter-task gradient conflicts through projection. When two task gradients have a negative inner product, it projects one onto the plane orthogonal to the other, then aggregates the adjusted gradients to update the shared model.
- **CAGrad:** CAGrad aims to minimize the average task loss while accounting for each task's local improvement. It searches within a constrained neighborhood of the average gradient for an update direction that maximizes the minimum local improvement across tasks.

- **FAMO:** FAMO dynamically adjusts task weights based on relative decreases in task losses to promote balanced optimization. In our experiments, tasks are sampled equally, while FAMO adjusts their contributions to the joint objective using task loss changes measured before and after each model update.
- **SEC:** SEC formulates task selection as a multi-armed bandit problem, using advantage statistics to estimate task learnability and dynamically adjust sampling probabilities. In our experiments, each task corresponds to an arm, whose value estimate is continually updated using the mean absolute advantage observed during training.
- **DADS:** DADS dynamically adjusts task sampling ratios based on historical rewards. It measures the degree of under-optimization through the gap between each task's smoothed reward and a target reward, assigning higher sampling probabilities to tasks with larger gaps.
- **DynaSample:** We construct DynaSample as an RPS-based dynamic sampling baseline. It uses the same remaining learning potential estimation as ProCore, converts task RPS values into sampling ratios through a softmax, and applies GRPO to the resulting mixed data. This baseline evaluates the use of RPS for task sampling without gradient cores enhancement.
- **MOPD:** We first train a separate teacher model for each task from the same initial model, then integrate their capabilities into a single student through multi-teacher on-policy distillation. During distillation, the student generates responses and receives token-level supervision from the teacher corresponding to each task. We adopt the policy-gradient distillation formulation presented in the original paper to update the student.

C.3 Training Details

We train Qwen3-4B and Qwen3-8B using the verl framework on four NVIDIA A100 GPUs. Table 10 summarizes the main training configurations for ProCore, including training hyperparameters, reward settings, and parameters governing gradient cores enhancement. Each training batch contains 256 prompts, with eight responses sampled per prompt, and each mini-batch contains 512 responses. Baseline-specific hyperparameters follow the settings reported in the respective original papers.

Table 10: Training hyperparameters and reward settings for ProCore.

Hyperparameter	Value
Learning rate	1×10^{-6}
Training batch size	256
Group size	8
Mini-batch size	512
Sampling temperature	1.0
Correctness reward	1.0
Max response length	1024
κ	1.1%
Step	120
τ	0.8
ρ	0.1

C.4 Main Results

In this section, we present the training reward curves of ProCore and Mixed Training on Qwen3-4B and Qwen3-8B, together with the changes in Remaining Potential Score (RPS) across tasks on Qwen3-8B, to further examine ProCore's performance during training.

Training Reward Dynamics. Figures 8 and 9 show the training reward curves for eight tasks at the two model scales, respectively. Compared with Mixed Training, ProCore achieves comparable or higher training rewards on the vast majority of tasks and a higher mean reward. These observations are consistent with the main experimental results and further support the effectiveness of ProCore across model scales.

Remaining Learning Potential Dynamics. Figure 10 shows the changes in RPS across tasks on Qwen3-8B. RPS generally decreases during training, but at different rates across tasks, reflecting differences in their remaining learning potential. For example, Bitwise Arithmetic reaches a relatively low RPS in the later stages of training, whereas Sokoban maintains a higher RPS. ProCore uses these signals to adjust the relative protection weights of tasks on shared gradient cores, assigning larger weights to tasks with higher RPS.

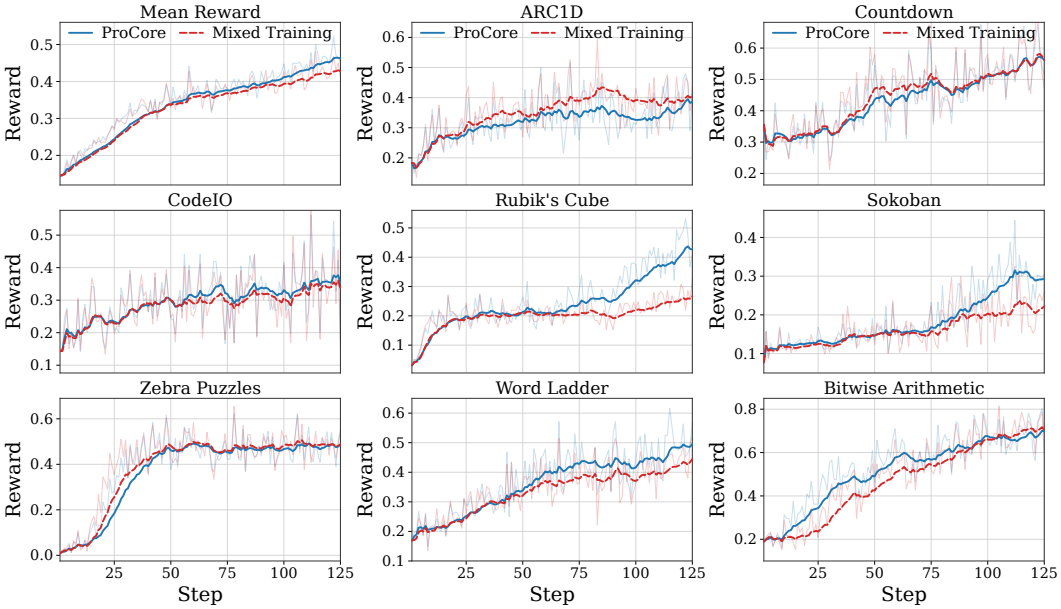


Figure 8: Reward curves of Mixed Training and ProCore on Qwen3-4B across eight tasks.

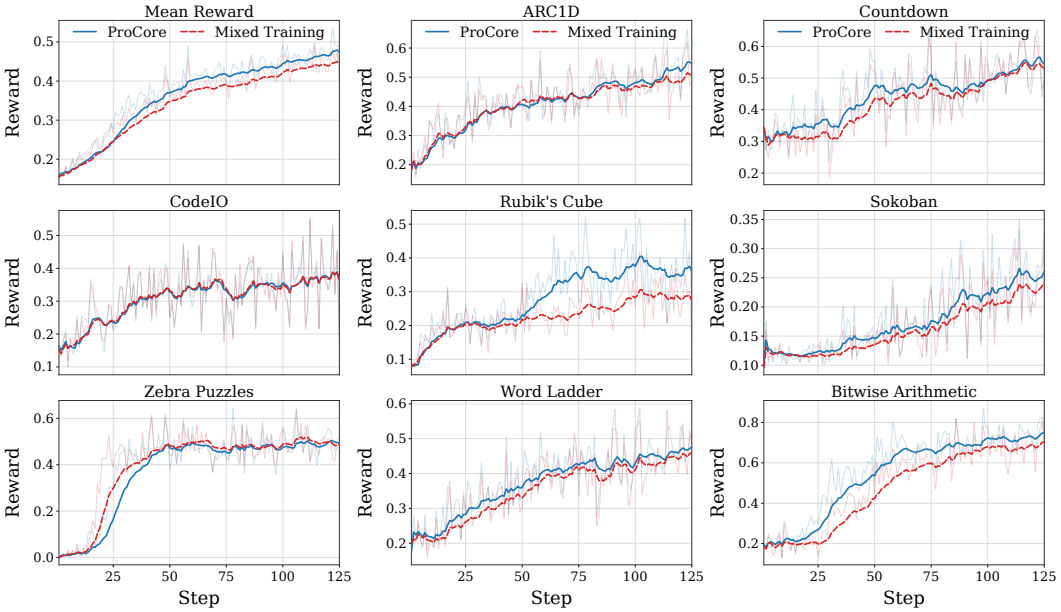


Figure 9: Reward curves of Mixed Training and ProCore on Qwen3-8B across eight tasks.

D Details of Generalization Experiments

D.1 Tasks and Datasets

To assess the generalizability of ProCore to broader tasks, we extend our experiments to three domains: mathematical reasoning, code generation, and scientific question answering. These domains

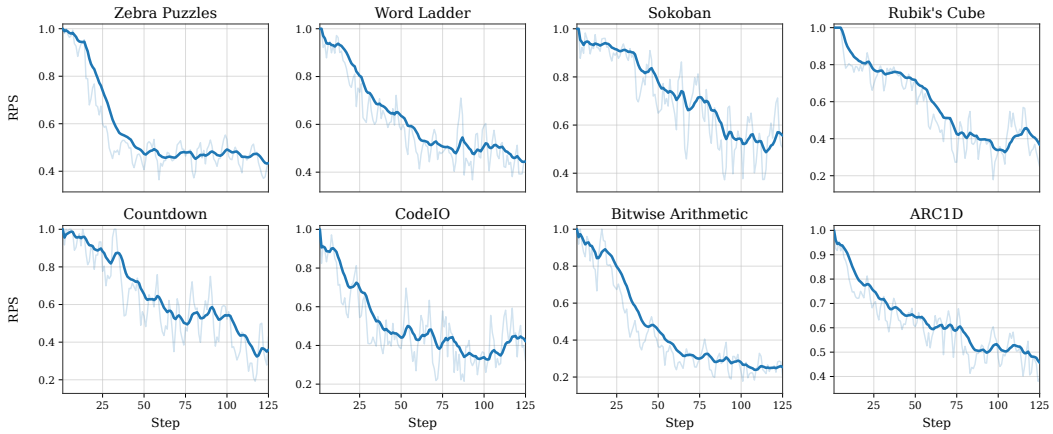


Figure 10: Changes in Remaining Potential Score across eight tasks on Qwen3-8B.

differ in problem format, answer space, and reward feedback: mathematical answers are primarily checked using rule-based verifiers, generated code is evaluated by executing test cases, and scientific answers are assessed for semantic consistency with reference answers. This setting allows us to examine the effectiveness of ProCore across different task formats and feedback mechanisms.

Training data. We construct training subsets for the three domains from Guru (Cheng et al., 2025). For mathematics and code generation, we first exclude samples with prompts exceeding 3,072 tokens, then perform stratified sampling based on the problem scores recorded in Guru. Each score p denotes the average reward over 16 responses generated by Qwen2.5-7B for the same problem.

- **Mathematical reasoning:** We initially select 6,250 problems from the mathematics portion of Guru. These include 1,250 problems with $p = 0$, along with 1,500, 2,000, and 1,500 problems sampled from the intervals $0 < p < 0.125$, $0.125 \leq p < 0.25$, and $0.25 \leq p < 0.375$, respectively, to cover a range of difficulty levels. After removing 27 problems confirmed to overlap with the evaluation sets, we retain 6,223 problems.
- **Code generation:** We select 4,740 problems from the code portion of Guru. These include 2,500 problems with $p = 0$, along with 1,090, 1,000, 100, and 50 problems sampled from the intervals $0 < p < 0.125$, $0.125 \leq p < 0.25$, $0.25 \leq p < 0.5$, and $p \geq 0.5$, respectively.
- **Scientific question answering:** We use the entire STEM subset of Guru, comprising 3,591 problems, retaining its full coverage of scientific question-answering tasks.

The three domains contain 14,554 training problems in total. All methods use the same training data to ensure consistent comparisons.

Evaluation data. We evaluate performance on nine benchmarks across mathematical reasoning, code generation, and scientific question answering, as detailed below.

- **Mathematical reasoning:** We use all 500 problems from MATH500 (Lightman et al., 2024), all 30 problems from AIME 2026 and all 33 problems from HMMT February 2026 provided by MathArena (Dekoninck et al., 2026), and 674 English competition mathematics problems from OlympiadBench (He et al., 2024). These benchmarks assess the models' ability to solve a range of mathematical problems.
- **Code generation:** We use HumanEval+ and MBPP+ from EvalPlus (Liu et al., 2023b), containing 164 and 378 programming problems, respectively, along with 175 programming problems from the time window corresponding to LiveCodeBench v6 (Jain et al., 2025). We execute the generated code and check its correctness using each benchmark's test cases. A solution is considered correct only if it passes all test cases.
- **Scientific question answering:** We use all 198 questions from GPQA-Diamond (Rein et al., 2023) and sample 200 questions from SuperGPQA (Du et al., 2026) using a fixed random seed of 42 to assess performance on challenging knowledge questions and cross-disciplinary reasoning tasks. The sampled evaluation questions are identical across all models and methods.

D.2 Training Details

We train Qwen3-1.7B, Qwen3-4B, and Qwen3-4B-Base using the verl framework. The method-specific hyperparameters of ProCore and all baselines remain the same as in the main experiments, as detailed in Appendix C.3. We use a learning rate of 1×10^{-6}. Each training batch contains 96 prompts, with eight responses sampled per prompt, and each mini-batch contains 192 responses. The maximum prompt and response lengths are both set to 4096 tokens. During training, we use a sampling temperature of 1.0 and top-p of 1.0, with rewards computed using domain-specific verification methods.

E Case Study

In this section, we present case studies of ProCore on eight datasets: ARC1D, Countdown, CodeIO, Rubik's Cube, Sokoban, Zebra Puzzles, Word Ladder, and Bitwise Arithmetic. Each case includes the task input, reasoning process, and final answer.

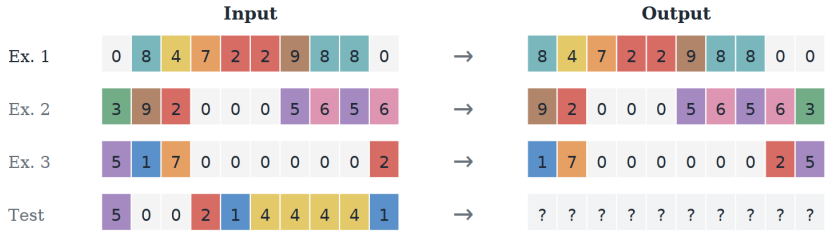
E.1 ARC1D

ARC1D

Infer a transformation from one-dimensional input-output examples.

Question

Find the common rule that maps an input grid to an output grid, given the examplesbelow. Predict the output for the test input and explain your reasoning.



Model response ProCore(Qwen3-4B)

To determine the rule that maps the input grid to the output grid, let's analyze the examples provided.
Example 1:
Input: 0 8 4 7 2 2 9 8 8 0
Output: 8 4 7 2 2 9 8 8 0 0
Observation: The output is the input shifted left by 1 position, with the first element (0) moved to the end.
Example 2:
Input: 3 9 2 0 0 0 5 6 5 6
Output: 9 2 0 0 0 5 6 5 6 3
Observation: Again, the output is the input shifted left by 1 position, with the first element (3) moved to the end.
Example 3:
Input: 5 1 7 0 0 0 0 0 0 2
Output: 1 7 0 0 0 0 0 0 2 5
Observation: Same pattern — the input is shifted left by 1, with the first element moved to the end.
Conclusion:
The rule is: **Shift all elements of the input list to the left by 1 position, and move the first element to the end of the list.**
Now, apply this rule to the test input:
Input: 5 0 0 2 1 4 4 4 4 1
Shift left by 1, move the first element (5) to the end:
Output: 0 0 2 1 4 4 4 4 1 5
Final Answer:

0 0 2 1 4 4 4 4 1 5

Figure 11: An example of ProCore on ARC1D.

E.2 COUNTDOWN

Countdown

Construct a target value while using each supplied number once.

Question

Using the numbers 31, 13, 87, 24, 76, create an equation that equals 5. Use each number exactly once and only the operations +, -, ×, ÷. Think step by step, then put the final expression inside a box.

Model response ProCore(Qwen3-4B)

I need to use the numbers [31, 13, 87, 24, 76] exactly once with basic arithmetic operations (+, -, ×, ÷) to create an equation that equals 5.

Let me try combinations:
First, try subtracting larger numbers:
87 - 76 = 11
31 - 24 = 7
11 + 7 = 18
18 - 13 = 5
So the equation is:
(87 - 76) + (31 - 24) - 13 = 5

Final Answer:

(87 - 76) + (31 - 24) - 13 = 5

Figure 12: An example of ProCore on Countdown.

E.3 CODEIO

CodeIO

Predict a program's output from its code and input.

Question

Predict the exact JSON output of the following program for the supplied input. It returns the 1-based line index that maximizes the second integer times the natural logarithm of the first.

```
import math
def main_solution(file_content):
    array = [line.split(",") for line in file_content.split("\n") if line]
    max_value = 0
    result = 0
    count = 1
    for i in array:
        if len(i) == 2:
            value = int(i[1]) * math.log(int(i[0]))
            if value > max_value:
                max_value = value
                result = count
        count += 1
    return result
```

Input

file_content = '37,32\n28,93\n15,23\n15,60\n36,38'

Model response ProCore(Qwen3-4B)

To determine the line number with the greatest product of the second integer and the natural logarithm of the first integer, we need to evaluate each line in the given file content.

The input is:
37,32
28,93
15,23
15,60
36,38

We calculate the value for each line as follows:
1. **Line 1:** 37,32
Value = 32 * ln(37) ≈ 32 * 3.6109 ≈ 115.55
2. **Line 2:** 28,93
Value = 93 * ln(28) ≈ 93 * 3.3322 ≈ 309.89
3. **Line 3:** 15,23
Value = 23 * ln(15) ≈ 23 * 2.7080 ≈ 62.28
4. **Line 4:** 15,60
Value = 60 * ln(15) ≈ 60 * 2.7080 ≈ 162.48
5. **Line 5:** 36,38
Value = 38 * ln(36) ≈ 38 * 3.5835 ≈ 136.17
The greatest value is approximately 309.89, which occurs on **line 2**.

Final Answer:

2

Figure 13: An example of ProCore on CodeIO.

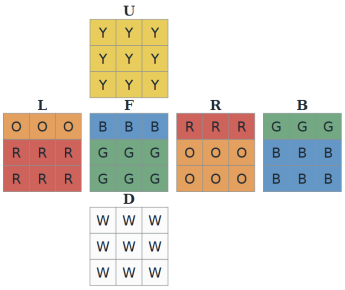
E.4 RUBIK'S CUBE

Rubik's Cube

Restore a scrambled cube with a valid sequence of face turns.

Question

This 3 × 3 × 3 cube was scrambled from solved with exactly 2 move(s). Return exactly 2 move(s) that solve it. Use Singmaster notation without combining moves (write U U instead of U2). The solution only uses moves on these face(s): U. Think step by step, then put only the final answer inside a box.



Model response ProCore(Qwen3-4B)

We are given a 3×3×3 Rubik's cube that was scrambled from the solved state with **exactly 2 moves**, and we are to return those 2 moves using **Singmaster notation**, **without combining moves** (e.g., "U2" is written as "U U").

Step 1: Understand the cube state

The cube is described as:

Y Y Y
Y Y Y
Y Y Y
O O O B B B R R R G G G
R R R G G G O O O B B B
R R R G G G O O O B B B
W W W
W W W
W W W

Let's interpret this:

- The **top face (U)** is all **Y** (Yellow).
- The **middle band of the net (L/F/R faces listed below)** has:
- First row: O O O B B B R R R
- Second row: R R R G G G O O O
- Third row: R R R G G G O O O
- The **bottom face (D)** is all **W** (White).

This configuration is not the solved state. It looks like the cube was rotated in some way.

Step 2: Determine the two moves that solve it

We are told the cube was solved and then scrambled with **exactly 2 moves**, and we are to return those 2 moves **using only moves on the U face**. After analyzing the cube state and testing possible U-face moves, the correct two moves that will return the cube to the solved state are:

- U
- U

These two moves (U followed by U) are equivalent to U2, but since we are not allowed to combine moves, we write them as separate moves.

Final Answer:

U U

Figure 14: An example of ProCore on Rubik's Cube.

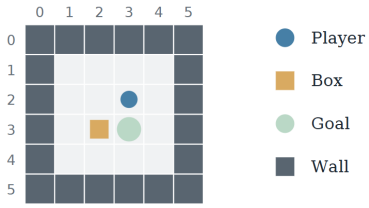
E.5 SOKOBAN

Sokoban

Plan player movements and pushes to place a box on its goal.

Question

Solve the Sokoban puzzle. The shortest solution is **exactly 4 moves**. Return a string of move characters using U, D, L, R. Think step by step, then put only the final answer inside a box.



Model response ProCore(Qwen3-4B)

Let's analyze the given Sokoban puzzle step by step.
Puzzle Layout:
Row 0: + + + + + +
Row 1: + - - - - +
Row 2: + - - * - +
Row 3: + - @ X - +
Row 4: + - - - - +
Row 5: + + + + + +
We can represent this as a grid:
Row 0: # # # # # #
Row 1: # - - - - #
Row 2: # - - * - #
Row 3: # - @ X - #
Row 4: # - - - - #
Row 5: # # # # # #

Key Elements:
- Walls: + (around the perimeter)
- Player: * at position (2,3)
- Box: @ at position (3,2)
- Goal: X at position (3,3)
- Empty space: -

Objective: Push the box (@) onto the goal (X).

Initial Position:
- Player is at (2,3)
- Box is at (3,2)
- Goal is at (3,3)

Plan: The player needs to move to the left of the box to push it right into the goal.
Let's simulate the shortest path:
1. Move Left (L): Player moves from (2,3) to (2,2)
2. Move Left (L): Player moves from (2,2) to (2,1)
3. Move Down (D): Player moves from (2,1) to (3,1)
4. Move Right (R): Player pushes the box from (3,2) to (3,3), which is a goal.
This results in the box being on the goal, completing the puzzle.

Final Answer:

LLDR

Figure 15: An example of ProCore on Sokoban.

E.6 ZEBRA PUZZLES

Zebra Puzzles

Combine relational clues to identify an attribute of a house.

Question

This is a logic puzzle. There are 5 houses (numbered 1 on the left, 5 on the right), from the perspective of someone standing across the street from them. Each has a different person in them. They have different characteristics:
- Each person has a unique name: carol, arnold, david, alice, bob
- People use different phone models: sony xperia 5, huawei p50, samsung galaxy s21, google pixel 6, one plus 9
- The people keep different animals: cat, horse, bird, fish, dog
- Each mother is accompanied by their child: fred, alice, bella, billy, timothy
- The people are of nationalities: brit, dane, german, mexican, chinese

1. The fish enthusiast is the person's child is named Alice.
2. The person who uses a Huawei P50 is the fish enthusiast.
3. The Mexican is directly left of the bird keeper.
4. Arnold and the person who is the mother of Timothy are next to each other.
5. The Chinese is Arnold.
6. The cat lover is directly left of the person who is the mother of Billy.
7. The Dane is in the second house.
8. The dog owner is in the fifth house.
9. The person who uses a Sony Xperia 5 is the person who is the mother of Billy.
10. Bob is in the fourth house.
11. The British person is the person who uses a Google Pixel 6.
12. Carol is the person who uses a OnePlus 9.
13. David is the person's child is named Fred.
14. The person who is the mother of Billy is Bob.
15. The person who uses a OnePlus 9 is the cat lover.

What is Animal of the person who lives in House 3? Provide only the answer as your final answer.

Model response ProCore(Qwen3-4B)

We are given a complex logic puzzle with 5 houses, each with unique attributes:
name, phone model, animal, child, and nationality.
We need to determine the **animal of the person in House 3**.
Step-by-step deductions:
From clue 10:
- Bob is in the fourth house.
From clue 14:
- The person who is the mother of Billy is Bob.
- So, Bob's child is Billy.
From clue 9:
- The person who uses a Sony Xperia 5 is the mother of Billy.
- So, Bob uses a Sony Xperia 5.
From clue 15:
- The person who uses a OnePlus 9 is the cat lover.
- From clue 12: Carol uses a OnePlus 9.
- So, Carol is the cat lover.
From clue 6:
- The cat lover is directly left of the person who is the mother of Billy.
- So, Carol is directly left of Bob.
- Since Bob is in House 4, Carol must be in House 3.
Therefore:
- House 3: Carol (cat lover)
Final Answer:

Cat

Figure 16: An example of ProCore on Zebra Puzzles.

E.7 WORD LADDER

Word Ladder

Connect two words through valid one-letter substitutions.

Question

Transform **PETS** into **CAMS** by changing exactly one letter at each step. Every word must be a valid English word. Return the sequence, including both endpoints, as uppercase words separated by commas without spaces.

Model response ProCore(Qwen3-4B)

We are given the word ladder: **PETS → CAMS**

We need to transform "PETS" into "CAMS" by changing one letter at a time, with each intermediate step being a valid English word.

Step 1: Start with "PETS"

We need to find a valid English word that differs from "PETS" by one letter.

Let's try changing each letter one at a time:

- P → C: "CETS" → Not a word
- E → A: "PATS" → Valid word
- T → M: "PEMS" → Not a word
- S → S: No change → Not helpful

So, "PATS" is a valid first step.

Step 2: From "PATS" to "CAMS"

Now we need to change one letter in "PATS" to get closer to "CAMS".

Try changing:

- P → C: "CATS" → Valid word
- A → A: No change
- T → M: "PAMS" → Not a word
- S → S: No change

So, "CATS" is a valid second step.

Step 3: From "CATS" to "CAMS"

Now we need to change one letter in "CATS" to get to "CAMS".

Try:

- C → C: No change
- A → A: No change
- T → M: "CAMS" → Valid word

Final Answer:

- PETS
- PATS
- CATS
- CAMS

Final Answer as a comma-separated sequence:

PETS,PATS,CATS,CAMS

Figure 17: An example of ProCore on Word Ladder.

E.8 BITWISE ARITHMETIC

Bitwise Arithmetic

Evaluate arithmetic and bitwise operations on signed integers.

Question

Evaluate the following expression and give the answer in hexadecimal. Assume arbitrary bit depth and signed integers. Show your reasoning

(((0x2e - 0x640)>> 0x3) - 0x34)

Model response ProCore(Qwen3-4B)

Let's solve the expression step by step:
((0x2e - 0x640) >> 0x3) - 0x34

Step 1: Convert the hexadecimal values to decimal
0x2e = 2 × 16 + 14 = 32 + 14 = 46
0x640 = 6 × 16^2 + 4 × 16 + 0 = 6 × 256 + 64 = 1536 + 64 = 1600
0x34 = 3 × 16 + 4 = 48 + 4 = 52

Step 2: Perform the subtraction 46 - 1600 = -1554

Step 3: Right shift by 0x3 = 3
-1554 >> 3
Right shifting a negative signed integer performs an arithmetic right shift. A right shift by 3 bits is equivalent to dividing by $2^3 = 8$ and taking the floor. Let's compute:
-1554 ÷ 8 = -194.25 ⇒ floor division = -195
So: -1554 >> 3 = -195

Step 4: Subtract 0x34 = 52
-195 - 52 = -247

Step 5: Convert -247 to hexadecimal
We convert 247 to hex: 247 ÷ 16 = 15 remainder 7
15 in hex is F
So 247_{10} = $0xF7_{16}$
Therefore, -247_{10} = $-0xF7_{16}$

Final Answer:

-0xf7

Figure 18: An example of ProCore on Bitwise Arithmetic.