TRAIN SMALL, GUIDE LARGE: OPTIMIZATION SCOUTING FOR EFFICIENT LLM POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) can substantially improve LLM reasoning, but its computational cost increases rapidly with model scale. This work investigates whether **low-cost RL on a smaller source model can replace full RL training for a larger target model**. We find that transferable optimization signals already emerge at relatively early stages of source-model RL and can be used to derive effective parameter updates without costly iterative optimization on the target model. Building on these findings, we propose **Optimization Scouting (OptScout)**, a framework for reusing RL-induced optimization signals across model scales. OptScout extracts evolving policy shifts during source-model RL and translates them into stable, target-specific update directions through low-rank gradient probing and cross-split consistency filtering. It then uses forward-only reward evaluation to select the best update direction and magnitude, and stops scouting once further source-model RL no longer improves transfer performance. Finally, the selected update is applied to the target model only once. Experiments across different source-side RL algorithms and target-model settings show that OptScout achieves higher average performance than direct target-side RL on multiple reasoning benchmarks, while providing up to a $6.32\times$ end-to-end speedup.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become an important approach for improving the reasoning capabilities of large language models (LLMs) (Shao et al., 2024; Guo et al., 2025; Hu et al., 2025; Gao et al., 2024; Yu et al., 2025). However, RLVR requires models to repeatedly generate long rollouts and perform multiple rounds of policy updates, making post-training increasingly expensive as model scale grows. Existing studies mainly improve RLVR efficiency through better policy optimization and sampling strategies, more efficient data selection, and faster rollout generation (Yu et al., 2025; Zheng et al., 2025; Li et al., 2025b; Wang et al., 2025). Despite these advances, scaling RLVR to larger models remains computationally expensive.

To further reduce the cost of RLVR post-training, we revisit RLVR from a functional perspective and view it as involving two core processes: **obtaining useful optimization information** and **turning it into parameter updates**. The former uses rollouts and reward feedback to identify which generation behaviors should be encouraged or discouraged, while the latter translates this information into parameter changes through backpropagation and repeated policy updates. Both processes become increasingly costly for larger models. This raises two questions: can useful optimization information be obtained at lower cost? And once such information is available, does turning it into effective parameter updates still require multiple rounds of optimization?

For obtaining optimization information, recent cross-scale studies have shown that RL-induced policy shifts in smaller models can provide useful guidance for larger models (Feng et al., 2026; Yu et al., 2026). We observe that stronger source-model performance does not necessarily lead to more effective cross-scale guidance. More importantly, useful cross-scale optimization information emerges early in source-model RL. policy shifts formed after only a few RL updates can already enable the target model to match or even surpass the performance of full RL training on the target model across multiple settings. Yet, even within this early stage, transfer utility varies noticeably across checkpoints, making it difficult to determine in advance which source state will provide the most effective guidance. Small models can therefore serve as low-cost optimization scouts: with only

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

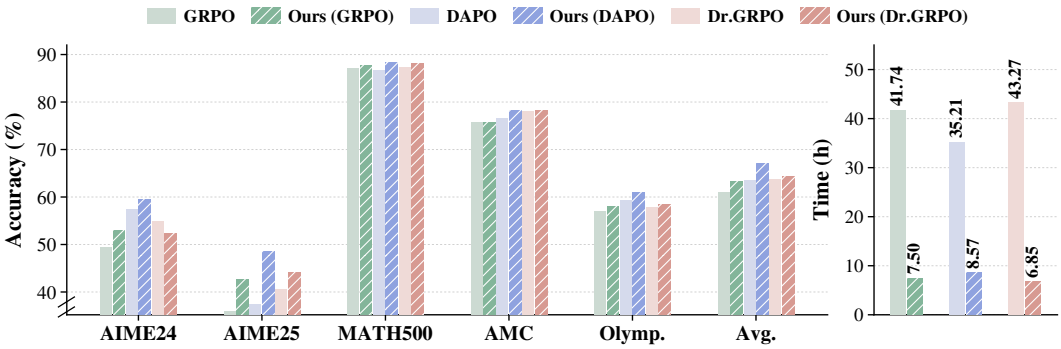

Figure 1: Main results on Qwen3-8B. Our method improves average performance across all three source-RL settings while substantially reducing post-training time.

a short phase of RL, they can reveal which generation behaviors RL begins to promote or suppress and provide effective optimization signals for larger target models.

Once such cross-scale optimization information is available, the next question is how to turn it into parameter updates for the target model. Token-level policy shifts indicate which generation behaviors should be promoted or suppressed, but do not directly specify an update direction in the target parameter space. A natural approach is to use these signals as training supervision and incorporate them through multiple rounds of backpropagation and parameter updates (Feng et al., 2026; Yu et al., 2026). However, this still retains the cost of iterative optimization on the larger model. We therefore ask whether the target model's own gradient response can be used to convert these cross-scale behavioral signals into an update direction in its parameter space, allowing the desired behavioral change to be achieved with a single calibrated parameter update. In this way, the target model can leverage optimization information from the smaller model without full target-side RL or iterative optimization, avoiding the need to continuously maintain the full training state of the target model and thereby reducing both computation and memory overhead.

Building on these findings, we propose **Optimization Scouting (OptScout)**, a framework for reusing RL-induced optimization signals across model scales. Rather than fixing in advance which source-policy state is used to construct the cross-scale signal, OptScout follows the evolution of the source policy during RL and treats the resulting policy shifts as candidate optimization signals. For each candidate, OptScout uses low-rank gradient probing on the target model to construct an update direction in its parameter space, while cross-split consistency filtering retains stable update components. It then uses forward-only generation and reward evaluation to select the best update direction and magnitude, and stops scouting once further source-model RL no longer yields a better candidate. Finally, the selected update is applied only once to the original target model, avoiding target-side RL and iterative optimization while enabling effective reuse of cross-scale optimization information at substantially lower target-side cost. Experiments across different source-side RL algorithms and target-model settings show that OptScout achieves higher average performance than direct target-side RL on multiple reasoning benchmarks, with up to a $6.32\times$ end-to-end post-training speedup and substantially lower peak memory usage.

The main contributions are summarized as follows:

- **Early emergence of cross-scale optimization information.** Useful cross-scale optimization information is shown to emerge early in source-model RL, while stronger source-model performance does not necessarily lead to more effective cross-scale guidance.
- **Target-space update construction.** A mechanism is introduced that uses the target model's own gradient response to convert RL-induced policy shifts into stable update directions in the target parameter space, without relying on target-side RL or iterative optimization.
- **Optimization Scouting.** OptScout unifies evolving policy shifts from a short phase of source-model RL, target-space update construction, and forward-only candidate selection into a single cross-scale post-training framework.

2 Related Work

2.1 Efficient Reinforcement Learning for LLM Post-Training.

Reinforcement learning has become an important paradigm for post-training large language models, with early work primarily focusing on aligning model behavior with human preferences (Ouyang et al., 2022; Dai et al., 2023). More recently, reinforcement learning with verifiable rewards (RLVR) has shown strong effectiveness in improving mathematical and general reasoning capabilities (Shao et al., 2024; Guo et al., 2025; Hu et al., 2025; Gao et al., 2024). As RLVR continues to scale, a growing body of work has focused on improving its stability and computational efficiency. Some methods improve policy optimization objectives or value estimation, including TBRM (Yuan et al., 2025), VinePPO (Kazemnejad et al., 2024), and VAPO (Yue et al., 2025); others improve sample efficiency through adaptive sampling, data filtering, resampling, or auxiliary training signals, including DAPO (Yu et al., 2025), GRESO (Zheng et al., 2025), RL-ZVP (Le et al., 2025), and AGPO (Li et al., 2025a); another line of work directly improves rollout-generation efficiency, including TreePO (Li et al., 2025c), PROS (Huang & Wan, 2026) and ARRoL (Xu et al., 2026).

2.2 Weak-to-Strong Generalization

Weak-to-strong generalization studies how supervision from weaker models can be used to improve stronger models (Burns et al., 2023), and has recently been extended to LLM reasoning (Yang et al., 2024; Yuan et al., 2026; Yao et al., 2025b;a). Recent work has explored using relative differences between weaker-models as supervision signals for stronger models. WSPO transfers the distributional change of a weak model before and after alignment to guide a stronger model (Zhu et al., 2025). Direct-OPD transfers RL-induced policy shifts from a smaller model to a stronger student through on-policy distillation (Feng et al., 2026). W2S-OPD extracts logit-level capability directions from contrasts between weaker models and transfers them via a proxy teacher and on-policy distillation (Yu et al., 2026). In contrast, OptScout explores candidate optimization signals along the evolving source-model RL trajectory rather than relying on a fixed source contrast. It uses target-model gradients to construct update directions and forward-only evaluation to select the final update, enabling early cross-scale transfer with lower target-side optimization cost.

3 Observations on Cross-Scale Optimization Signals

Recent studies have shown that policy shifts induced by RL in smaller models can provide useful cross-scale guidance for larger target models (Feng et al., 2026; Yu et al., 2026). A natural question is: **when does such transferable optimization information emerge during source-model RL?** Intuitively, as the source model undergoes more RL updates and achieves better task performance, its policy shifts might be expected to provide increasingly effective guidance for the target model. Our results do not support this assumption.

3.1 Cross-Scale Guidance Emerges Early in Source RL

We first investigate when useful cross-scale optimization information emerges during source-model RL. We train a 1.7B source model with GRPO and Dr.GRPO and divide the RL process into three stages: Early (steps 5–25), Middle (steps 50–70), and Late (steps 95–115), with five checkpoints in each stage. For each stage, we evaluate both the source model itself and the 8B target model updated using the corresponding policy shifts. All results are measured on MATH500 using mean@16. As shown in Figure 2, source-model performance continues to improve substantially as RL proceeds. From the Early to Late stage, the 1.7B source model improves by 3.0 and 2.2 percentage points under GRPO and Dr.GRPO,

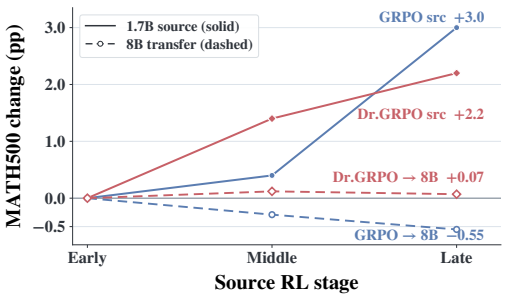

Figure 2: Relative changes in source and transfer performance across RL stages.

respectively. In contrast, the corresponding cross-scale transfer performance does not improve accordingly: the 8B transfer performance decreases by 0.55 points under GRPO and increases by only 0.07 points under Dr.GRPO. This reveals a clear mismatch between source-model performance and cross-scale transfer effectiveness. Although additional RL updates continue to improve the source model itself, they do not consistently produce stronger optimization signals for the larger target model. These results suggest that useful cross-scale optimization information already emerges in the early stage of source-model RL, before the source model reaches its stronger later-stage performance.

3.2 Transfer Effectiveness Varies Across Early Checkpoints

The previous subsection shows that useful cross-scale optimization information already emerges in the early stage of source-model RL. However, this does not imply that a particular early checkpoint can be fixed in advance as the source of the optimization signal. We therefore examine successive checkpoints from the early portion of source-model RL. As shown in Table 1, transfer effectiveness varies noticeably across successive source checkpoints. The 8B target reaches 87.0 at step 10, drops to 85.4 at step 15, and then rises again to 87.2 at step 20. Further source RL does not lead to consistent improvement, with performance reaching 86.4 and 86.6 at steps 25 and 30, respectively. These results show that, although useful cross-scale optimization signals emerge early, their effectiveness can still vary substantially across checkpoints. Therefore, the training step alone does not reliably indicate which source checkpoint will provide the most effective cross-scale guidance. This observation motivates Optimization Scouting. Rather than relying on a predetermined early checkpoint, we treat policy shifts emerging throughout source-model RL as candidate optimization signals and directly evaluate their effect on the target model. This allows OptScout to identify a more effective source signal without waiting for the completion of source-model RL.

Table 1: 8B transfer performance across early source checkpoints.

Source RL step	5	10	15	20	25	30
MATH500 (%)	84.8	87.0	85.4	87.2	86.4	86.6

4 Method

4.1 Overview

We consider a smaller source model and a larger target model. Our goal is to use the smaller model to reveal useful optimization signals at low cost and efficiently translate them into parameter updates for the larger target model. Let π_S^0 and π_S^k denote the initial source policy and the source policy after k RL steps, respectively, and let π_T^0 and θ_0 denote the initial target policy and its parameters. OptScout uses policy shifts that emerge during source-model RL as cross-scale optimization signals. For each source checkpoint, we extract token-level policy shifts on trajectories generated by the target model, use the target model's own low-rank gradient response to convert them into candidate update directions, and retain parameter coordinates consistently supported across data splits. We then evaluate candidates associated with different source checkpoints and update magnitudes using forward generation and verifiable rewards, and apply only the best candidate once to the target model.

4.2 Policy Shifts on Target Trajectories

To characterize the policy changes induced by source-model RL, we first use the initial target policy π_T^0 to generate a fixed set of trajectories. We denote the i-th question–response pair by (x_i, y_i), where x_i is the input question and $y_i = (y_{i,1}, \ldots, y_{i,L_i})$ is the target-generated response of length L_i. For the source checkpoint π_S^k obtained after k RL steps, we evaluate the initial source policy π_S^0 and the current source policy π_S^k under teacher forcing on the same target-generated sequence y_i, and compute the conditional log-probability of each token. Their difference characterizes the token-level policy change induced by k steps of source-model RL relative to the initial policy, and is further processed to construct the target-side optimization signal. Specifically, for the t-th token in the i-th response, we define the raw policy shift as

$$\Delta_{i,t} = \log \pi_S^k \left(y_{i,t} \mid x_i, y_{i,<t}\right) - \log \pi_S^0 \left(y_{i,t} \mid x_i, y_{i,<t}\right), \qquad t = 1, \ldots, L_i. \tag{1}$$

The sign of $\Delta_{i,t}$ indicates whether source-model RL increases or decreases the probability of the token relative to the initial source policy. To reduce the influence of extreme shifts, we clip the

log-probability differences and then standardize them globally:

$$w_{i,t} = \frac{\mathrm{clip}\left(\Delta_{i,t}, -c, c\right) - \mu_c}{\max\left(\sigma_c, \epsilon\right)}, \tag{2}$$

where c is the clipping threshold, μ_c and σ_c are the mean and standard deviation of the clipped policy shifts over all response tokens at the current source checkpoint, and $\epsilon > 0$ is a small constant for numerical stability. The resulting $w_{i,t}$ serves as a signed shift weight. Note that $\Delta_{i,t}$ describes how source-model RL changes the token probability, whereas the centered $w_{i,t}$ reflects the position of this shift relative to the overall shift distribution. Therefore, their signs do not necessarily coincide. Let $\pi_T(\cdot;\theta)$ denote the target policy parameterized by θ. The shift-weighted objective is then defined as:

$$\mathcal{L}_{\mathrm{shift}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \frac{\sum_t w_{i,t}\left[-\log \pi_T\left(y_{i,t} \mid x_i, y_{i,<t}; \theta\right)\right]}{\max\left(\sum_t |w_{i,t}|, 1\right)}. \tag{3}$$

Here, N is the number of target trajectories. Global standardization controls the distribution of token-level shift weights, while per-trajectory normalization prevents trajectories with larger total shifts from dominating the gradient response. The sign of $w_{i,t}$ determines whether the local update tends to increase or decrease the target model's probability of the corresponding token. Because the shift weights are signed, $\mathcal{L}_{\mathrm{shift}}$ is not used as a conventional likelihood-training objective. Instead, it serves as a shift-weighted target-side objective whose local gradient response at the original target parameters θ_0 will be used to construct the update direction.

4.3 Target-Space Update Directions

Policy shifts describe how generation behavior should change, but they do not directly specify a parameter update for the target model. We therefore use low-rank gradient probing to obtain the local gradient response of $\mathcal{L}_{\mathrm{shift}}$, and stabilize it with cross-split consistency filtering. For notational simplicity, we suppress the checkpoint superscript k on intermediate quantities in this subsection.

Low-Rank Gradient Probing. To obtain the local gradient response of a larger target model without explicitly computing and storing its full gradients, we perform gradient probing through a low-rank parameterization. Specifically, for a linear weight matrix $W \in \mathbb{R}^{m\times n}$, we temporarily introduce a rank-r parameterization:

$$W' = W + \alpha B_W A_W, \tag{4}$$

where $B_W \in \mathbb{R}^{m\times r}$, $A_W \in \mathbb{R}^{r\times n}$, and α is the effective scaling factor used by the probe. For each W, A_W is fixed as the probe basis and B_W is initialized to zero. All probe gradients are evaluated at the original target parameters θ_0 with $B_W = 0$. Based on $\mathcal{L}_{\mathrm{shift}}$ defined in the previous subsection, we compute its gradient with respect to B_W and map the resulting low-rank response back to the original parameter space through a ridge-stabilized transformation:

$$\widetilde{G}_W = -\frac{1}{\alpha}\nabla_{B_W}\mathcal{L}_{\mathrm{shift}}\left(A_W A_W^\top + \lambda_W I_r\right)^{-1} A_W. \tag{5}$$

Here, $\widetilde{G}_W \in \mathbb{R}^{m\times n}$ has the same shape as W, and I_r denotes the $r \times r$ identity matrix. It is not the full gradient, but a ridge-stabilized parameter-space response derived from the rank-r probe gradient. The scale-adaptive damping term λ_W improves the numerical stability of inverting $A_W A_W^\top$ and suppresses response amplification caused by poor conditioning. We apply the probe to the q/k/v/o attention projections and the gate/up/down MLP projections in each Transformer layer; these matrices constitute the eligible parameter set used in the following consistency filtering step. The detailed derivation and the choice of λ_W are provided in Appendix A.1.

Cross-Split Consistency Filtering. To reduce noise in the target-space responses $\widetilde{G}_W$ and retain stable update components, we evaluate their consistency across two prompt-disjoint trajectory subsets. Specifically, we partition the target trajectories by prompt into two subsets and compute the probe response separately on each subset. The two splits use the same fixed probe basis A_W and probe configuration, differing only in the data used to compute the gradients. This gives two response

matrices, $\widetilde{G}_W^{(1)}$ and $\widetilde{G}_W^{(2)}$, for each W. To identify parameter coordinates that receive consistent responses from the two splits, we compare the two responses at the scalar-coordinate level. We flatten the responses from all eligible attention and MLP projection matrices and perform global filtering over their scalar coordinates. Let $\mathcal{E}$ denote the resulting set of scalar coordinates. For coordinate j, $\widetilde{g}_j^{(1)}$ and $\widetilde{g}_j^{(2)}$ denote its corresponding entries in $\widetilde{G}_W^{(1)}$ and $\widetilde{G}_W^{(2)}$, where W is the weight matrix containing that coordinate. To account for scale differences across weight matrices, we normalize each probe response by the root-mean-square (RMS) magnitude of its corresponding weight matrix and define the cross-split consistency score:

$$\hat{g}_j^{(a)} = \frac{\widetilde{g}_j^{(a)}}{\mathrm{RMS}(W)}, \quad a \in \{1,2\}, \qquad s_j = \sqrt{\max\left(\hat{g}_j^{(1)}\hat{g}_j^{(2)}, 0\right)}. \tag{6}$$

If the two splits assign opposite signs to a coordinate, then $s_j = 0$. When their signs agree, the geometric mean of the two relative response magnitudes measures their shared support, so a weak response from either split lowers the score. We use the consistency scores s_j to select a sparse set of parameter coordinates that are strongly and consistently supported by both splits. Given a retention ratio ρ, we retain at most $K = \lceil \rho|\mathcal{E}| \rceil$ coordinates. Specifically, we rank all coordinates in $\mathcal{E}$ by s_j and retain up to the K highest-scoring coordinates with $s_j > 0$. Let $\mathcal{S}$ denote the selected set. For each selected coordinate, we average the two matrix-relative responses, $\bar{d}_j = \frac{1}{2}\left(\hat{g}_j^{(1)} + \hat{g}_j^{(2)}\right)$. We then normalize the selected coordinates by their global RMS and set unselected coordinates to zero:

$$d_j^k = \begin{cases} \dfrac{\bar{d}_j}{\sqrt{\frac{1}{|\mathcal{S}|}\sum_{\ell \in \mathcal{S}} \bar{d}_\ell^2}}, & j \in \mathcal{S}, \\ 0, & j \notin \mathcal{S}. \end{cases} \tag{7}$$

The resulting sparse direction $\mathbf{d}^k$ is used as the candidate update direction associated with source policy π_S^k. The matrix-wise and global RMS normalizations make responses across weight matrices and directions across source checkpoints comparable in scale.

4.4 Forward-Only Candidate Evaluation and One-Time Update

For each source checkpoint π_S^k, we obtain a corresponding candidate update direction $\mathbf{d}^k$. OptScout uses forward evaluation to select an appropriate update magnitude and compare candidate updates produced by different source checkpoints. We first construct a target-specific hard set $\mathcal{D}_{\mathrm{hard}}$ from a fixed reranking pool. Specifically, the initial target policy π_T^0 generates responses on this pool, and questions that it fails to solve are retained. The resulting $\mathcal{D}_{\mathrm{hard}}$ remains fixed throughout scouting and is disjoint from all final test benchmarks. Let $\eta > 0$ denote the candidate update magnitude. For a weight matrix W, the candidate parameter change is defined as

$$\delta W_{k,\eta} = \eta \ \mathrm{RMS}(W)\, d_W^k, \tag{8}$$

where d_W^k denotes the part of $\mathbf{d}^k$ corresponding to W. The factor $\mathrm{RMS}(W)$ restores the scale of the corresponding weight matrix, while η controls the update magnitude along the candidate direction. For each source checkpoint, we instantiate candidate models using a predefined set of update magnitudes η, always starting from the same initial parameters θ_0. We evaluate these candidates on $\mathcal{D}_{\mathrm{hard}}$ by their average verifiable reward and retain the best update magnitude for that checkpoint. As source RL proceeds, OptScout compares the best candidate from each checkpoint and stops source training when no candidate exceeds the best score observed so far for P consecutive scouting checkpoints. Finally, we select the highest-scoring checkpoint–magnitude pair among all evaluated candidates and apply its corresponding parameter change once to the original target model θ_0.

5 Experiments

5.1 Experimental Setup

Models and Data. We use Qwen3-1.7B as the source model and Qwen3-8B as the primary target model, and additionally evaluate R1-Distill-7B as a cross-family target and Qwen3-14B as a larger-scale target. The source model is trained with GRPO, DAPO, and Dr.GRPO. The training set

Table 2: Main results on Qwen3-8B. All benchmark results are reported in percentage (%). Avg. denotes the macro average over the five benchmarks. For each RL algorithm, Δ denotes the difference between our cross-scale method and directly applying the corresponding RL algorithm to Qwen3-8B.

Method	AIME24	AIME25	MATH500	AMC	Olympiad	Avg.	Time ↓
Qwen3-8B	41.25	23.96	80.28	61.97	45.84	50.66	–
Direct-OPD	48.54	35.21	83.96	71.99	50.66	58.07	28.01 h
GRPO	49.38	35.83	87.14	75.75	56.97	61.01	41.74 h
Ours	**52.92**	**42.71**	**87.65**	**75.68**	**57.90**	**63.37**	**7.50 h**
Δ vs. GRPO	+3.54	+6.88	+0.51	−0.07	+0.93	+2.36	5.57×
Dr.GRPO	54.79	40.63	87.35	77.94	57.76	63.69	43.27 h
Ours	**52.29**	**44.17**	**88.18**	**78.16**	**58.38**	**64.24**	**6.85 h**
Δ vs. Dr.GRPO	−2.50	+3.54	+0.83	+0.22	+0.62	+0.55	6.32×
DAPO	57.29	37.29	86.59	76.58	59.18	63.39	35.21 h
Ours	**59.38**	**48.54**	**88.38**	**78.24**	**60.94**	**67.10**	**8.57 h**
Δ vs. DAPO	+2.09	+11.25	+1.79	+1.66	+1.76	+3.71	4.11×

Table 3: Results across different target models. All benchmark results are reported in percentage (%). Avg. denotes the macro average over five benchmarks.

Target	Method	AIME24	AIME25	MATH500	AMC	Olympiad	Avg.	Time ↓
R1-Distill-7B	Base	38.75	30.42	80.45	70.63	47.32	53.51	–
	GRPO	43.75	32.92	83.15	76.36	52.36	57.71	32.00 h
	Ours	**47.92**	**40.42**	**84.53**	**75.30**	**53.88**	**60.41**	**5.80 h**
	Δ	+4.17	+7.50	+1.38	−1.06	+1.52	+2.70	5.52×
Qwen3-14B	Base	37.71	30.21	84.24	66.87	49.81	53.77	–
	GRPO	58.13	46.25	89.13	80.57	61.69	67.15	66.49 h
	Ours	**64.17**	**47.50**	**89.43**	**80.65**	**61.19**	**68.59**	**18.58 h**
	Δ	+6.04	+1.25	+0.30	+0.08	−0.50	+1.44	3.58×

contains 1,500 mathematical reasoning examples constructed from MATH (Hendrycks et al., 2021) and DeepScaleR (Luo et al., 2025), and the direct target-side RL baselines use the same training data.

Baselines and Evaluation. On Qwen3-8B, we compare OptScout with the base model, direct GRPO (Shao et al., 2024), DAPO (Yu et al., 2025), and Dr.GRPO (Liu et al., 2025), and the cross-scale baseline Direct-OPD. For Direct-OPD, we select the best-performing source-RL checkpoint as the teacher. On R1-Distill-7B and Qwen3-14B, we compare with the corresponding base model and direct GRPO. We evaluate on AIME 2024 (Art of Problem Solving, 2026a), AIME 2025 (Art of Problem Solving, 2026a), MATH500 (Hendrycks et al., 2021), AMC 2023 (Art of Problem Solving, 2026b), and OlympiadBench (He et al., 2024), sampling 16 responses per problem and reporting verifier-based mean@16 and its macro-average across benchmarks.

Implementation Details. For GRPO and Dr.GRPO, OptScout performs scouting every five source-RL steps, while DAPO uses a denser two-step interval due to its dynamic sampling. Starting from the fourth scouting round, source RL stops after two consecutive non-improving scouting checkpoints. For wall-clock evaluation, Qwen3-14B uses eight Ascend 910 NPUs, while Qwen3-8B and R1-Distill-7B use eight NVIDIA GeForce RTX 5090 GPUs; methods using the same target model use the same hardware. OptScout time is measured end-to-end from the start of source RL through the final update, whereas direct target-side RL is measured from the start of training to the reported checkpoint. Full hyperparameters and implementation details are provided in the Appendix A.2.

5.2 Main Results

Table 2 presents the main results on Qwen3-8B. With GRPO, Dr.GRPO, and DAPO as the source-side RL algorithms, OptScout consistently achieves higher average performance than direct target-side RL

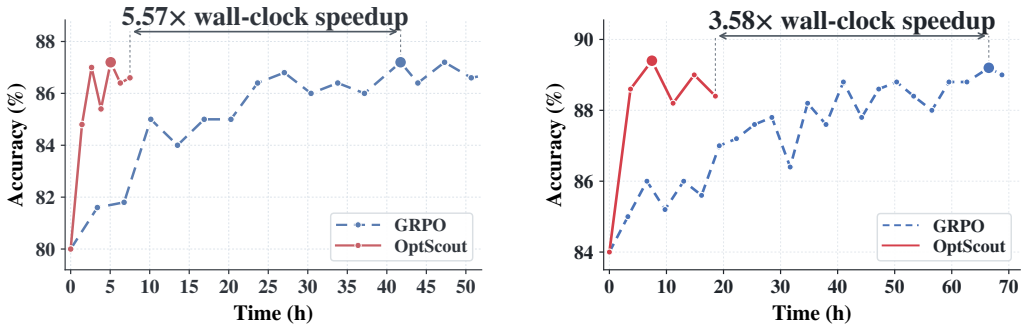

Figure 3: Post-training dynamics on MATH500 for Qwen3-8B (left) and Qwen3-14B (right).

while reducing end-to-end post-training time by $4.11\times$–$6.32\times$. Among these settings, OptScout with DAPO achieves the highest average score of 67.10. Compared with the cross-scale baseline Direct-OPD, OptScout also achieves higher average performance with lower post-training cost. These results show that OptScout can effectively reuse cross-scale optimization signals across different source-side RL algorithms. Table 3 further evaluates OptScout along two dimensions: generalization to a target model outside the Qwen3 family and scaling to a larger target model. For these settings, we use DeepSeek-R1-Distill-Qwen-7B (R1-Distill-7B) and Qwen3-14B as the target models, respectively. In both settings, OptScout achieves higher average performance than direct GRPO, together with $5.52\times$ and $3.58\times$ end-to-end speedups, respectively. These results show that the performance and efficiency gains of OptScout extend beyond the primary setting to other target-model settings. Additional out-of-domain results on MBPP are provided in Appendix A.3.

5.3 Efficiency Analysis

Runtime Efficiency. Figure 3 shows MATH500 performance as a function of wall-clock time for Qwen3-8B and Qwen3-14B. Compared with direct GRPO, the reported OptScout models are obtained with substantially less post-training time, corresponding to $5.57\times$ and $3.58\times$ wall-clock speedup on Qwen3-8B and Qwen3-14B, respectively. The efficiency advantage remains substantial as the target model scales from 8B to 14B. Table 4 further breaks down the end-to-end runtime of OptScout under the $1.7\text{B} \rightarrow 8\text{B}$ setting. Most of the runtime is spent on small-model RL and forward-only candidate evaluation, while policy-shift acquisition and direction construction account for relatively small portions of the total cost. This indicates that extracting cross-scale optimization signals and converting them into update directions does not become a major computational bottleneck.

Table 4: Wall-clock breakdown of OptScout for $1.7\text{B} \rightarrow 8\text{B}$.

Stage	Time	Share
Source-side		
Source RL	3.18 h	42.4%
Target-side		
Policy shift	0.54 h	7.2%
Direction	0.70 h	9.3%
Candidate eval.	2.82 h	37.6%
Update/export	0.26 h	3.5%
Total	**7.50 h**	**100%**

Memory Efficiency. We further compare peak GPU memory under a separate controlled setting, where all methods use $8\times$ NVIDIA GeForce RTX 5090 GPUs, each exposing 31.843 GiB of GPU memory under our system configuration. As shown in Table 5, on Qwen3-8B, OptScout reduces peak per-GPU memory usage by approximately 25.3% compared with GRPO and Direct-OPD. Under the same memory constraint, the complete OptScout pipeline remains feasible on Qwen3-14B, whereas both direct GRPO and Direct-OPD run out of memory. This advantage comes from keeping iterative optimization on the smaller source model, while the target model only performs low-rank gradient probing and forward-only candidate evaluation. Detailed stage-wise measurements and configurations are provided in the Appendix A.4.1.

Table 5: Peak per-GPU memory.

Method	8B	14B
GRPO	31.83 GiB	OOM
Direct-OPD	31.84 GiB	OOM
OptScout	**23.78 GiB**	**24.07 GiB**

Table 6: Ablation of different policy-shift signals on Qwen3-8B.

Signal Variant	AIME24	AIME25	MATH500	AMC	Olympiad	Avg.
Qwen3-8B	41.25	23.96	80.28	61.97	45.84	50.66
Random	38.13	25.63	80.13	61.30	45.94	50.23
Token-shuffle	2.71	1.88	51.90	25.23	20.16	20.38
Positive-only	51.46	39.79	86.73	74.17	56.68	61.77
Negative-only	46.46	34.17	87.23	78.61	57.48	60.79
Full Shift (Ours)	**52.92**	**42.71**	**87.65**	**75.68**	**57.90**	**63.37**

Table 7: Ablation of target-space direction construction on Qwen3-8B. "Eligible Params." denotes the attention and MLP projection matrices used by OptScout.

Direction Variant	AIME24	AIME25	MATH500	AMC	Olympiad	Avg.
Full Gradient (All Params.)	37.50	30.42	81.71	64.46	48.13	52.44
Full Gradient (Eligible Params.)	50.00	32.71	86.88	71.61	55.16	59.27
Low-Rank Probe (Eligible Params.)	**56.04**	36.88	87.10	71.69	56.96	61.73
Low-Rank + Magnitude Top-2%	49.17	40.00	85.80	73.34	56.25	60.91
Low-Rank + Cross-Split Top-2%	52.92	**42.71**	**87.65**	**75.68**	**57.90**	**63.37**

5.4 Ablation Studies

Policy-Shift Signals. We first study how different policy-shift signals affect cross-scale transfer. As shown in Table 6, random signals yield an average score of 50.23 on Qwen3-8B, comparable to the base model at 50.66, while shuffling the correspondence between tokens and their policy shifts causes the score to drop substantially to 20.38. This shows that the transfer gains depend on the token-level policy shifts induced by source-model RL rather than arbitrary token weighting. Using only positive or negative shifts achieves average scores of 61.77 and 60.79, respectively, while the full signed shifts further improve the score to 63.37, indicating that both promoted and suppressed generation behaviors provide useful optimization information.

Target-Space Direction Construction. We next analyze how these policy shifts are translated into parameter updates for the target model. As shown in Table 7, the dense full-gradient response over all model parameters achieves an average score of 52.44, while restricting it to the eligible attention and MLP projection matrices improves the score to 59.27, showing that the choice of target parameter subset is important for cross-scale transfer. On the same eligible parameter set, the low-rank probe without sparsification further reaches 61.73. Selecting the Top-2% coordinates solely by response magnitude gives 60.91, whereas cross-split consistency filtering reaches 63.37, indicating that agreement across data splits provides a more effective criterion for sparse coordinate selection than magnitude alone. Furthermore, compared with the non-sparse low-rank direction over all eligible parameters, Cross-Split Top-2% reduces the serialized update-direction size from 13.89 GB to 1.67 GB, an approximately 88% reduction, while also improving average performance. Additional hyperparameter sensitivity results are reported in Appendix A.5.

6 Conclusion

This work revisits LLM reinforcement learning post-training from two perspectives: **obtaining optimization information** and **translating it into parameter changes**. We find that policy shifts emerging early in small-model RL already provide effective cross-scale optimization information, and that this information can be converted into effective parameter updates through the target model's own gradient response. Building on these findings, we propose OptScout, which obtains optimization signals through short source-model RL and realizes cross-scale transfer with a single target-model update. Experiments show that OptScout achieves higher average reasoning performance while substantially reducing post-training cost and peak memory usage.

AI USE STATEMENT

We used generative AI tools to improve writing clarity and language, to summarize and analyze relevant literature, and to provide suggestions on the organization and structure of the paper. The research questions, methodology, experimental design, and interpretation of the results were developed independently by the authors. All AI-assisted outputs were manually reviewed, verified, and revised by the authors. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work focuses on improving the computational efficiency of LLM post-training and does not involve human subjects or private user data. The experiments use established research models and publicly available benchmarks. We do not identify specific ethical concerns beyond those generally associated with the development and deployment of large language models, and appropriate evaluation and safeguards should be considered for real-world applications.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce the main experiments in the paper and appendix. Section 5 describes the models, datasets, baselines, evaluation protocol, hardware, and wall-clock measurement procedure. Appendix A.1 provides the derivation and implementation details of the low-rank gradient probe, while Appendix A.2 reports the training configurations, OptScout hyperparameters, data construction, candidate selection, and final evaluation protocol. Additional experimental details, controlled memory measurements, and hyperparameter sensitivity analyses are provided in the appendix. All compared methods use the same final benchmark evaluation protocol.

REFERENCES

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2026a. Accessed: 2026-03-07.

Art of Problem Solving. Amc problems and solutions. https://artofproblemsolving.com/wiki/index.php/AMC_Problems_and_Solutions, 2026b. Accessed: 2026-03-07.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. Safe rlhf: Safe reinforcement learning from human feedback. *arXiv preprint arXiv:2310.12773*, 2023.

Shiyuan Feng, Huan-ang Gao, Haohan Chi, Hanlin Wu, Zhilong Zhang, Zheng Jiang, Bingxiang He, Wei-Ying Ma, Ya-Qin Zhang, and Hao Zhou. Weak-to-strong generalization via direct on-policy distillation. *arXiv preprint arXiv:2607.05394*, 2026.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

Baizhou Huang and Xiaojun Wan. Pros: Towards compute-efficient rlvr via rollout prefix reuse. In *The Fourteenth International Conference on Learning Representations*, 2026.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. 2024.

Thanh-Long V Le, Myeongho Jeon, Kim Vu, Viet Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in llm reinforcement learning via entropy-guided advantage shaping. *arXiv preprint arXiv:2509.21880*, 2025.

Chen Li, Nazhou Liu, and Kai Yang. Adaptive group policy optimization: Towards stable training and token-efficient reasoning. *arXiv preprint arXiv:2503.15952*, 2025a.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Limr: Less is more for rl scaling. *arXiv preprint arXiv:2502.11886*, 2025b.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, et al. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling. *arXiv preprint arXiv:2508.17445*, 2025c.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 3(5), 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.

Haobo Xu, Sirui Chen, Ruizhong Qiu, Yuchen Yan, Chen Luo, Monica Xiao Cheng, Jingrui He, and Hanghang Tong. Prune as you generate: Online rollout pruning for faster and better rlvr. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13876–13893, 2026.

Yuqing Yang, Yan Ma, and Pengfei Liu. Weak-to-strong reasoning. In *Findings of the association for computational linguistics: EMNLP 2024*, pp. 8350–8367, 2024.

Wei Yao, Gengze Xu, Huayi Tang, Wenkai Yang, Donglin Di, Ziqiao Wang, and Yong Liu. On weak-to-strong generalization and f-divergence. *arXiv preprint arXiv:2506.03109*, 2025a.

Wei Yao, Wenkai Yang, Ziqiao Wang, Yankai Lin, and Yong Liu. Revisiting weak-to-strong generalization in theory and practice: Reverse kl vs. forward kl. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2860–2888, 2025b.

Fangxu Yu, Weijia Xu, Michael Xu, Tianyi Zhou, and Zinan Lin. Weak-to-strong on-policy distillation. *arXiv preprint arXiv:2607.26246*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yige Yuan, Teng Xiao, Shuchang Tao, Xue Wang, Jinyang Gao, Bolin Ding, and Bingbing Xu. Incentivizing strong reasoning from weak supervision. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7138–7156, 2026.

Yurun Yuan, Fan Chen, Zeyu Jia, Alexander Rakhlin, and Tengyang Xie. Trajectory bellman residual minimization: A simple value-based method for llm reasoning. *arXiv preprint arXiv:2505.15311*, 2025.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Haizhong Zheng, Yang Zhou, Brian R Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *arXiv preprint arXiv:2506.02177*, 2025.

Wenhong Zhu, Zhiwei He, Xiaofeng Wang, Pengfei Liu, and Rui Wang. Weak-to-strong preference optimization: Stealing reward from weak aligned model. In *International Conference on Learning Representations*, volume 2025, pp. 37116–37144, 2025.

A Appendix

A.1 Method Details and Derivations

This appendix gives the derivation of the low-rank gradient probe used in Eq. (5) and specifies the damping coefficient used in our implementation. The derivation also makes explicit the parameter-space geometry of the probe. Throughout this section, $W \in \mathbb{R}^{m\times n}$ denotes one eligible attention or MLP projection matrix, r is the probe rank, and

$$G_W = \nabla_{W'} \mathcal{L}_{\text{shift}}|_{B_W=0} = \nabla_W \mathcal{L}_{\text{shift}} \tag{9}$$

denotes the gradient evaluated at the original target parameters θ_0.

A.1.1 Derivation of Low-Rank Gradient Probing

For each eligible matrix W, we temporarily parameterize the weight as

$$W' = W + \alpha B_W A_W, \qquad B_W \in \mathbb{R}^{m\times r}, \quad A_W \in \mathbb{R}^{r\times n}, \tag{10}$$

where A_W is a fixed probe basis, B_W is initialized to zero, and α is the effective LoRA scaling factor. In our implementation, A_W is the LoRA-A factor created by the standard PEFT initialization: it uses the default Kaiming-uniform initialization and is then kept fixed throughout gradient probing; the experiment seed determines its realized subspace. The same realized A_W is used for the two prompt-disjoint data splits. Consequently, the forward model at the probing point is unchanged, since $W' = W$ when $B_W = 0$. Let $\langle X, Y\rangle_F = \operatorname{tr}(X^\top Y)$ denote the Frobenius inner product. A perturbation of B_W gives

$$dW' = \alpha\, dB_W A_W. \tag{11}$$

Using the differential of the shift-weighted objective,

$$\begin{aligned} d\mathcal{L}_{\text{shift}} &= \langle G_W, dW' \rangle_F \\ &= \alpha \langle G_W, dB_W A_W \rangle_F \\ &= \alpha \left\langle G_W A_W^\top, dB_W \right\rangle_F . \end{aligned} \tag{12}$$

we obtain the chain-rule relation

$$\nabla_{B_W} \mathcal{L}_{\text{shift}} = \alpha G_W A_W^\top. \tag{13}$$

This relation is evaluated at $B_W = 0$; hence it uses the same forward point as the original target model and does not require training or retaining a LoRA adapter.

Substituting Eq. (13) into the reconstruction rule in Eq. (5) gives

$$\begin{aligned} \widetilde{G}_W &= -\frac{1}{\alpha} \left(\alpha G_W A_W^\top\right) \left(A_W A_W^\top + \lambda_W I_r\right)^{-1} A_W \\ &= -G_W A_W^\top \left(A_W A_W^\top + \lambda_W I_r\right)^{-1} A_W. \end{aligned} \tag{14}$$

The response has shape $m \times n$, the same as W. Since the ridge operator is positive semidefinite for $\lambda_W \geq 0$, the leading minus sign makes $\widetilde{G}_W$ a descent direction whenever the response is nonzero. Its orientation and magnitude are determined jointly by the shift-weighted gradient response and the probe subspace. The sparse support used for the final parameter update is selected later by cross-split consistency filtering and the global top-ρ selection described in the main text.

When $\lambda_W = 0$ and A_W has full row rank, $A_W A_W^\top$ is invertible and Eq. (14) reduces to

$$\widetilde{G}_W = -G_W A_W^\top (A_W A_W^\top)^{-1} A_W = -G_W \Pi_{A_W}, \qquad \Pi_{A_W} = A_W^\top (A_W A_W^\top)^{-1} A_W. \tag{15}$$

Here Π_{A_W} is the orthogonal projector onto the row space of A_W in $\mathbb{R}^n$. Thus, each row of the full gradient is projected onto the probe basis row space. The low-rank probe therefore retains precisely the part of the full gradient that is observable through the rank-r perturbation; it is a structured low-rank measurement rather than an independent LoRA training step.

For $\lambda_W > 0$, define the ridge operator

$$\Pi_{A_W, \lambda_W} = A_W^\top (A_W A_W^\top + \lambda_W I_r)^{-1} A_W. \tag{16}$$

If $A_W = U \Sigma V^\top$ is a compact singular value decomposition, then

$$\Pi_{A_W, \lambda_W} = V \ \text{diag}\left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda_W}\right) V^\top. \tag{17}$$

This operator provides a ridge-regularized reconstruction of the gradient response. It approaches the orthogonal projector as $\lambda_W \to 0$, while directions associated with small singular values receive smaller weights. This regularization improves numerical stability when $A_W A_W^\top$ is ill-conditioned. For $\lambda_W > 0$, it is generally not idempotent and should therefore be understood as a spectral shrinkage operator rather than an exact projector. In the implementation, the product in Eq. (16) is evaluated with a linear solve rather than by explicitly forming a matrix inverse.

A.1.2 Choice of the Damping Coefficient

We use a scale-adaptive ridge coefficient for every eligible matrix W. Let

$$\bar{d}_W = \frac{1}{r} \operatorname{tr}(A_W A_W^\top) = \frac{1}{r} \|A_W\|_F^2. \tag{18}$$

The coefficient used by the implementation is exactly

$$\lambda_W = \max\left(10^{-8},\ 10^{-6}\, \bar{d}_W\right). \tag{19}$$

Equivalently, the code computes the mean of the diagonal of $A_W A_W^\top$ and multiplies it by 10^{-6}, followed by the numerical floor 10^{-8}. The factor 10^{-6} is therefore relative to the scale of the probe Gram matrix, rather than a fixed absolute regularizer. The floor is the only additional numerical

safeguard in λ_W; the ϵ used elsewhere when standardizing token-level shift weights is a separate constant and is not part of this damping rule.

The same rule is applied independently to every eligible projection matrix and every Transformer layer. Thus, all layers share the same multiplier and floor, while λ_W can differ across matrices when their probe bases have different norms. The probe configuration uses the fixed rank specified in the main text (rank $r = 8$ in our experiments). With the standard PEFT scaling used by the implementation ($use_rslora = False$), $lora_alpha = 16$ gives the effective scale

$$\alpha = \frac{lora_alpha}{r} = \frac{16}{8} = 2. \tag{20}$$

Thus, α in Eq. (10) denotes the effective scale 2, rather than the raw configuration value `lora_alpha` $= 16$; this scale is divided out in Eq. (14). Thus, λ_W is used only to stabilize the reconstruction of the probe response and does not affect how the probe gradient $\nabla_{B_W} \mathcal{L}_{\text{shift}}$ itself is computed.

A.1.3 Response Normalization for Cross-Split Filtering

For completeness, the RMS normalization used by the cross-split consistency filter in Eq. (6) is defined for each eligible weight matrix $W \in \mathbb{R}^{m \times n}$ as

$$\mathrm{RMS}(W) = \sqrt{\frac{1}{mn} \sum_{u=1}^{m} \sum_{v=1}^{n} W_{uv}^2} = \frac{\|W\|_F}{\sqrt{mn}}. \tag{21}$$

For split $a \in \{1, 2\}$ and scalar coordinate j belonging to W, the relative response is therefore

$$\widehat{g}_j^{(a)} = \frac{\widetilde{g}_j^{(a)}}{\mathrm{RMS}(W)}. \tag{22}$$

In the implementation, $\mathrm{RMS}(W)$ is computed from the original base weight in FP32, and the denominator is replaced by $\max(\mathrm{RMS}(W), 10^{-12})$ for numerical safety. This normalization makes response magnitudes comparable across matrices with different weight scales; the subsequent cross-split score and global top-ρ selection then determine the sparse update support.

A.2 Detailed Experimental Setup

This section details the training data, RL configurations, OptScout settings, and final evaluation protocol. Source-side RL and direct target-side RL use the same training pool. Generation settings are specified separately for RL training, direction construction, candidate scoring, and final evaluation. All methods use the same protocol for final benchmark evaluation.

A.2.1 Models and Data

Table 8 summarizes the model and data choices. The source policy is Qwen3-1.7B, with Qwen3-8B as the primary target. We additionally evaluate DeepSeek-R1-Distill-Qwen-7B (R1-Distill-7B) and Qwen3-14B to study cross-family generalization and transfer to a larger target model, respectively. Examples are consumed in a fixed order with shuffling disabled; no further question-level deduplication is applied after mixing. Thus, repeated prompts that occur as separate source rows retain their original row identifiers. When a prompt-disjoint split is needed for direction extraction, normalized duplicate prompts are assigned to the same split so that they cannot cross the split boundary.

The training and evaluation prompts use the standard step-by-step reasoning instruction and the benchmark-required final-answer format. Overlong prompts are handled according to the training configuration. For each benchmark, the same prompts are used across all compared methods.

For each target model, we use a fixed 500-example reranking subset drawn from the same 1,500-example training manifest. The initial target policy is evaluated on this pool using one sampled rollout per example with temperature 0.7, top-$p = 0.95$, and a maximum response length of 8,192 tokens. The questions whose sampled responses fail verification form the target-specific hard set

Table 8: Models, data pools, and fixed preprocessing choices.

Item	Configuration
Source policy	Qwen3-1.7B.
Target policies	Qwen3-8B, R1-Distill-7B, and Qwen3-14B.
Training set	1,500 mathematical reasoning examples constructed from MATH and DeepScaleR.
Training order and deduplication	Fixed example order; no post-mix question-level deduplication. Normalized duplicate prompts are kept in the same cross-split group.
Reranking pool	Fixed 500-example subset drawn from the training set.
Final benchmarks	AIME 2024, AIME 2025, MATH500, AMC 2023, and OlympiadBench; all are held out from training and reranking.

Table 9: RL configurations used for source and direct target training.

Setting	GRPO	DAPO	Dr.GRPO
Source / target models	Qwen3-1.7B / Qwen3-8B, R1-Distill-7B, and 14B	Qwen3-1.7B / Qwen3-8B	Qwen3-1.7B / Qwen3-8B
Learning rate	1×10^{-6}	1×10^{-6}	1×10^{-6}
Prompt batch size	64	64	64
Rollouts per prompt	8	8	8
PPO mini-batch	32	32 (dynamic batching)	32
Prompt length	2,048	2,048	2,048
Response length	8,192	8,192	8,192
Temperature	1.0	1.0	1.0
Top-p	1.0	1.0	1.0
KL regularization	Disabled	Disabled	Disabled
Optimizer	AdamW; weight decay 0.01; 0 warm-up steps	AdamW; weight decay 0.1; 10 warm-up steps	AdamW; weight decay 0.01; 0 warm-up steps
Precision	bfloat16	bfloat16	bfloat16
Advantage rule	Standard deviation normalization	DAPO group filtering and clipping	No standard deviation normalization

D_{hard}. Consequently, $|D_{\mathrm{hard}}|$ can differ across target sizes, while its membership is fixed before candidate reranking. The 500-row pool may also be used by the source RL run, but it is never used as a final benchmark score.

The held-out benchmarks are separate from the training and reranking pools. We perform normalized exact-prompt overlap checks between the training/reranking manifests and all five final evaluation sets, with no overlaps found. For each benchmark, the answer-extraction and verification rules are held fixed across all methods.

A.2.2 RL Training Configuration

Table 9 summarizes the RL hyperparameters. The corresponding OptScout source runs use the same algorithmic settings as their source-side baselines; their duration is determined by the early-stopping rule given below.

For the Direct-OPD cross-scale baseline, the pre-RL Qwen3-1.7B model is used as the reference, and the best-performing source-RL checkpoint is selected as the teacher. The reference and teacher are then kept fixed while the Qwen3-8B target student is optimized with the Direct-OPD on-policy distillation objective from its base checkpoint. Except for the teacher checkpoint selection described above, we follow the official Direct-OPD implementation and its default hyperparameter settings.

For the direct-RL baseline on R1-Distill-7B, we use the same GRPO configuration as for Qwen3-8B; only the target-model checkpoint is changed. R1-Distill-7B has 28 decoder layers, while the remaining GRPO hyperparameters, rollout settings, optimizer, precision, and evaluation protocol are unchanged.

Table 10: OptScout direction extraction, sparsification, and candidate selection.

Setting	Configuration
Direction trajectories N	100 target trajectories per source-RL checkpoint.
Shift clipping c	2.
Probe rank r	8.
Probe scale	$\mathrm{lora_alpha} = 16$ with rank $r = 8$, hence the effective $\alpha = 2$.
Probe initialization	Standard PEFT Kaiming-uniform A_W, zero B_W; A_W fixed and shared across the two halves.
Probe modules	All-layer query, key, value, and output attention projections, together with gate, up, and down MLP projections; embeddings, the language-model head, normalization parameters, and biases are excluded.
Target layers	36 layers for Qwen3-8B, 28 layers for R1-Distill-7B, and 40 layers for Qwen3-14B; the probe covers layers 0 through $L - 1$.
Split	50/50 prompt-disjoint split by normalized prompt group.
Direction decoding	One rollout, temperature 0.7, top-p 0.95, max response 8,192, max model length 16,384.
Sparsity ρ	Global Top-2% after cross-split consistency scoring.
Scouting interval	Every 5 source-RL steps for GRPO/Dr.GRPO; every 2 source-RL steps for DAPO.
Candidate magnitudes	$\eta \in \{0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.008, 0.01\}$ for all target models.
Candidate population	Eight candidates per source checkpoint (one per η), using the trajectory direction.
Candidate scoring	One deterministic rollout per example in D_{hard}, with at most 8,192 response tokens; select the highest average verifiable reward across checkpoints and magnitudes.

A.2.3 OptScout Configuration

The main OptScout configuration is summarized in Table 10. For a source checkpoint k and magnitude η, candidate scoring computes the average verifiable reward on the fixed D_{hard} set using forward generation only. No optimizer update is performed during candidate evaluation.

For GRPO and Dr.GRPO, scouting is performed every five source-RL steps. DAPO uses a two-step interval to match its dynamic sampling and token-batching schedule. Starting from the fourth scouting round, we apply early stopping: source RL terminates if $P = 2$ consecutive scouting checkpoints fail to improve the best candidate score observed so far. The highest-scoring candidate among all evaluated checkpoints and magnitudes is selected, and its update is applied once to the original target model.

A.2.4 Evaluation Protocol

For a benchmark problem q, let $z_{q,k}$ be the verifier score of the k-th independently decoded response, with $k \in \{1, \ldots, 16\}$. The reported mean@16 is

$$\mathrm{mean@16}(q) = \frac{1}{16} \sum_{k=1}^{16} \mathbf{1}\{z_{q,k} > 0\}, \qquad \mathrm{mean@16}(D) = \frac{1}{|D|} \sum_{q \in D} \mathrm{mean@16}(q).$$

The final number in the main table is the macro-average of the five benchmark means (AIME 2024, AIME 2025, MATH500, AMC 2023, and OlympiadBench), so each benchmark contributes one equally weighted score regardless of its number of problems.

All methods use the common vLLM evaluator and decoding protocol in Table 11. Candidate reranking is a separate deterministic procedure; its one-response score is not the final mean@16 metric.

A.3 Out-of-Domain Evaluation on MBPP

We further evaluate OptScout on MBPP (Austin et al., 2021), a code-generation benchmark, to examine its behavior beyond the mathematical reasoning benchmarks used in the main experiments. We use the standard 374-example training split and 500-example test split. For OptScout, the training split is partitioned into 100 examples for trajectory probing and 274 disjoint examples for candidate

Table 11: Final benchmark evaluation protocol.

Item	Configuration
Benchmarks	AIME 2024, AIME 2025, MATH500, AMC 2023, OlympiadBench.
Responses per problem	16 independently decoded responses.
Decoding	Temperature 1.0, top-$p = 1.0$, top-$k = -1$, with sampling enabled.
Prompt length	2,048 tokens.
Response length	8,192 tokens.
Verifier	Benchmark-specific answer extraction and comparison with the reference answer; fixed across all methods.
Per-benchmark metric	Mean of the 16-response correctness fraction over problems.
Overall metric	Unweighted macro-average over the five benchmark means.
Evaluator parity	Identical evaluator, prompt handlers, decoding parameters, and verifier for every method.

scoring. For direct target-side GRPO, we use the full 374-example training split, while all other training hyperparameters follow the same configuration as in the mathematical reasoning experiments. The held-out test split is excluded from source-side RL, trajectory probing, candidate scoring, and model selection.

Table 12: Results on MBPP using Qwen3-8B.

Method	**MBPP**	**Time**
GRPO	84.66	26.23 h
OptScout	**84.96**	**4.88 h**
Δ	+0.30	$5.38\times$

For final evaluation, both direct target-side GRPO and OptScout use the same evaluator and decoding configuration. We generate 16 responses for each of the 500 test problems with temperature 1.0, top-$p = 1.0$, and a maximum response length of 8,192 tokens, and report mean@16 by averaging the binary execution rewards across the 16 responses for each problem and then across all test problems. As shown in Table 12, OptScout achieves 84.96 on MBPP, comparable to 84.66 obtained by direct target-side GRPO, while reducing the end-to-end post-training time from 26.23 hours to 4.88 hours, corresponding to a $5.38\times$ wall-clock speedup. These results show that OptScout maintains performance comparable to direct RL beyond mathematical reasoning while substantially reducing post-training cost.

A.4 Controlled Memory Analysis

A.4.1 Memory-Measurement Configuration

We use the same model checkpoints, training manifest, and fixed reranking pool as in Appendix A.2. Memory profiles are collected on one node with eight NVIDIA GeForce RTX 5090 GPUs. NVML is sampled every 0.1 s on all visible devices, and the maximum used memory over devices and sampling times is recorded. The idle baseline is recorded before each run for reference but is not subtracted from the reported value. The reported memory statistic is the raw NVML used-memory value, including non-PyTorch allocations.

For the 8B comparison, GRPO is profiled for five target-RL steps and Direct-OPD for five target-side optimization steps, while OptScout is profiled for five source-RL steps followed by one target update. For Qwen3-14B, the OptScout profile uses five source-RL steps followed by one target update; GRPO and Direct-OPD are run under the same controlled configuration until they encounter out-of-memory (OOM). GRPO and Direct-OPD use a training batch of 64 and eight rollouts per prompt; OptScout uses the same source-side settings. The OptScout profile includes the fixed hard-pool rollout, target rollout, direction construction, magnitude reranking, and one-shot update/export stages. It uses 100 direction trajectories per source checkpoint, 500 fixed hard-pool questions, eight magnitude candidates, rank $r = 8$, cross-split Global Top-2% selection, 36 target layers for Qwen3-8B, and 40 target layers for Qwen3-14B.

All memory profiles use the same prompt and response limits, tensor and sequence parallelism, rollout concurrency, vLLM controls, precision, offloading settings, token budgets, and 0.1-s NVML sampling interval. The configuration is therefore held fixed across model sizes and workloads; only the model, workload type, and profiled training steps differ.

Table 13: Configuration of the controlled GPU-memory measurements.

Setting	Configuration
Hardware	Eight NVIDIA GeForce RTX 5090 GPUs on one node; 31.84 GiB reported capacity per card.
Prompt / response limits	2,048 / 8,192 tokens; maximum model length 10,240.
Rollout parallelism	Tensor parallel size 4 and sequence-parallel size 8.
Rollout concurrency	$rollout_max_num_seqs = 1024$.
vLLM controls	$gpu_memory_utilization = 0.70$; eager execution enabled; prefix caching and chunked prefill disabled.
Training precision	FP32 model initialization with bfloat16 FSDP computation.
Memory controls	Parameter, optimizer, reference-model, and activation offload enabled where supported; gradient checkpointing, cache-engine freeing, and actor forward prefetch enabled.
Token budgets	PPO, rollout-log-prob, and reference-log-prob budgets of 1,280 tokens per GPU; PPO micro-batch size 1 per GPU.

Table 14: Peak per-GPU memory usage across different stages of OptScout.

Stage	**Qwen3-8B (GiB)**	**Qwen3-14B (GiB)**
Hard pool	23.70	23.81
Source RL	23.78	23.78
Target rollout	22.93	23.00
Policy shift	5.60	5.60
Direction	12.12	19.22
Rerank	23.68	23.78
One-shot update	23.48	**24.07**

A.4.2 Stage-Wise Memory Analysis

We record stage-wise GPU memory usage under a controlled setting with $8\times$ NVIDIA GeForce RTX 5090 GPUs (31.843 GiB/GPU). Figure 4 shows representative memory traces for OptScout, GRPO, and Direct-OPD. On Qwen3-8B, the peak per-GPU memory usage is 23.78 GiB for OptScout, 31.83 GiB for GRPO, and 31.84 GiB for Direct-OPD. On Qwen3-14B, the complete OptScout pipeline remains executable under the same memory constraint, with a peak of 24.07 GiB, whereas GRPO and Direct-OPD run out of memory and therefore do not have complete 14B traces. Table 14 provides a stage-wise breakdown for OptScout. For Qwen3-8B, the highest memory usage occurs during Source RL at 23.78 GiB, while for Qwen3-14B the peak occurs during One-shot update at 24.07 GiB. Hard pool, Target rollout, and Rerank remain around 23–24 GiB for both model sizes, while Policy shift uses 5.60 GiB. Memory usage during Direction increases from 12.12 GiB on Qwen3-8B to 19.22 GiB on Qwen3-14B, but remains below the 32 GiB device capacity.

A.5 Sensitivity Analysis

We further study the sensitivity of OptScout to the retention ratio ρ and the number of probing trajectories N. All other settings are kept fixed at their default values.

All main experiments use $\rho = 2\%$ and $N = 100$. Across the tested retention ratios, MATH500 mean@16 remains within 87.09–87.65. In particular, increasing ρ from 2% to 10% changes the result only slightly from 87.65 to 87.60, while a substantially larger ratio of 20% still achieves 87.36. This indicates that the method is not highly sensitive to the exact sparsity level within the tested range, and that retaining a small fraction of consistently supported coordinates is already sufficient to obtain strong transfer performance.

The number of probing trajectories has a clearer effect when the probing set is small. Increasing N from 25 to 50 improves MATH500 mean@16 from 85.91 to 86.78, and increasing it further to 100 raises the score to 87.65. However, using 200 trajectories yields only a marginal additional improvement, from 87.65 to 87.68. This trend suggests that additional probing trajectories mainly help by stabilizing the estimated target-side update direction when N is small, whereas the benefit

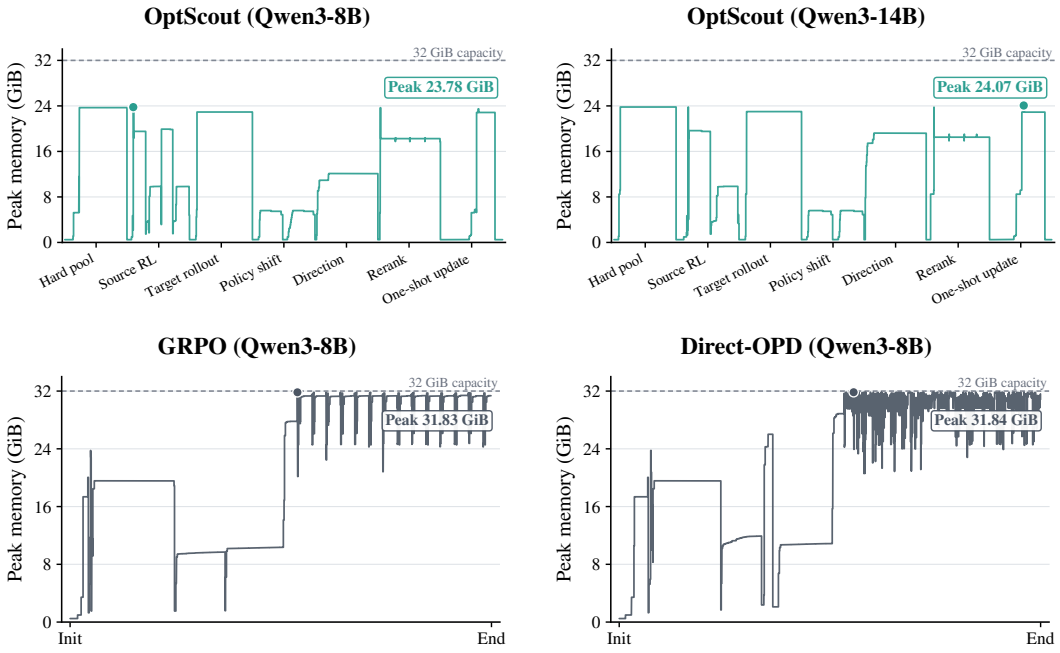

Figure 4: Peak GPU memory traces under the controlled $8\times$ RTX 5090 setting with 32 GiB per GPU. The top row shows OptScout on Qwen3-8B and Qwen3-14B, while the bottom row shows GRPO and Direct-OPD on Qwen3-8B. GRPO and Direct-OPD are omitted for Qwen3-14B because both run out of memory under this setting.

Table 15: Sensitivity to the retention ratio and the number of probing trajectories on Qwen3-8B. Results are MATH500 mean@16 (%).

Retention Ratio		**Probing Trajectories**	
ρ	**MATH500**	N	**MATH500**
1%	87.09	25	85.91
2%	**87.65**	50	86.78
10%	87.60	100	87.65
20%	87.36	200	**87.68**

begins to saturate around $N = 100$. We therefore use $N = 100$ in the main experiments as a practical balance between transfer performance and the additional cost of target-side probing.