REWARD-ALIGNED DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards improves the reasoning capabilities of large language models, but remains computationally expensive and sample inefficient. On-policy distillation (OPD) provides dense token-level supervision from a stronger teacher, yet standard OPD distills every token regardless of whether the teacher's preference actually improves task reward. Existing token-selection methods rely on proxies such as uncertainty or teacher–student divergence, which need not identify reward-improving supervision. We introduce Reward-Aligned Distillation (RAD), a bilevel framework that weights distillation signals based on their predicted effect on the student's post-update reward. RAD differentiates through a temporary optimizer step to score each token's supervision, then updates the token weights by projected ascent from the initial token weights. The resulting weights can strengthen, suppress, or reverse individual distillation-gradient contributions. At a matched training-response budget, RAD improves mean accuracy on six competition-math benchmarks over the strongest evaluated baselines by 6.5 percentage points for Qwen3-1.7B (non-thinking) and 2.5 points for Qwen3-4B-Instruct-2507. On LiveCodeBench-v6, RAD reaches 54.69% and 58.67% pass@1 with Qwen3-1.7B (non-thinking) and OLMo3-7B-Instruct, respectively, exceeding the strongest evaluated baselines by 8.7 and 6.4 points at the matched training-response budget. These results support reward-guided weighting of teacher supervision across model families.

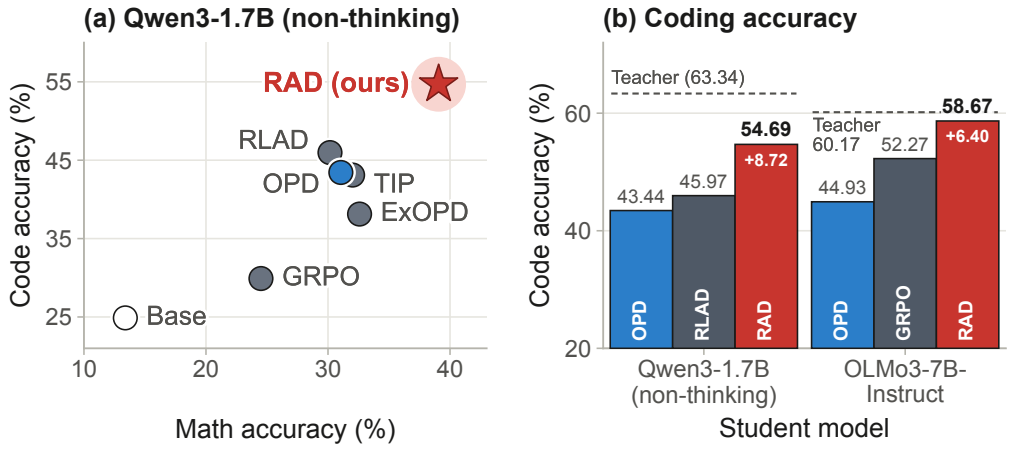

Figure 1: Reward-aligned distillation (RAD) consistently outperforms OPD and other baselines on mathematical reasoning and code generation tasks. The teachers are Qwen3-30B-A3B-Instruct-2507 for the Qwen student and OLMo3.1-32B-Instruct for the OLMo student. (a) Rollout-matched Qwen3-1.7B (non-thinking) results for methods with both math and coding evaluations (Tables 1 and 8). (b) LiveCodeBench-v6 pass@1 for Qwen3-1.7B (non-thinking) and OLMo3-7B-Instruct, comparing RAD at step 100 with baselines at step 200. The grey bars show the strongest evaluated baselines, RLAD and GRPO, respectively; each RAD bar gives its gain over that baseline (pp). Dashed lines mark the corresponding teachers.

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.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has been shown effective for improving mathematical reasoning (Luong & Lockhart, 2025) and code generation (Jaech et al., 2024; Ekbote et al., 2025) abilities, but its supervision is sparse: a long response often receives only a single sequence-level reward (Shao et al., 2024; Guo et al., 2025). On-policy distillation offers a complementary signal by evaluating student-generated responses under a strong teacher and supplying dense token-level guidance (Agarwal et al., 2024; Gu et al., 2024). This approach is appealing because the student continues to explore on policy while learning from the teacher at every generated token.

Standard OPD, however, implicitly treats the teacher as the target policy and every teacher preference as worth imitating. This would be justified if the teacher were optimal for the target reward, but a stronger teacher need not be reward-optimal. Teacher–student differences may reflect style, formatting, verbosity, or reasoning choices that do not improve task reward (Armandpour et al., 2026). Consequently, reducing teacher–student divergence does not guarantee reward improvement: uniform distillation can spend capacity matching reward-irrelevant behavior or move the student away from reward-improving solutions (Xu et al., 2026; Akhondzadeh et al., 2026).

The challenge is therefore to determine which teacher signals will benefit the student's task performance. Token-selection methods (Xu et al., 2026; Xiao et al., 2026) use heuristic criteria such as uncertainty, teacher–student disagreement, and reasoning structure to selectively apply teacher supervision. These criteria identify potentially useful tokens, but do not directly assess whether the corresponding distillation update improves the student policy's task reward. We therefore seek to choose each token's distillation weight based on its predicted effect on reward: *retain teacher signals that help the student solve the task, and suppress or reverse those that hinder it.*

To this end, we introduce reward-aligned distillation (RAD), which weights teacher supervision by its predicted effect on task reward. Token weights affect reward indirectly, by controlling how the student learns from teacher supervision. RAD captures this dependency with a bilevel objective: the inner problem trains the student with weighted distillation, while the outer problem selects weights to maximize the trained student's reward (Equation 2). We approximate the inner optimization with one temporary optimizer step using token weights initialized to a tunable value. Differentiating the temporary student's reward with respect to these weights yields look-ahead scores that guide a projected ascent update. The resulting signed weights can strengthen, suppress, or reverse individual distillation-gradient contributions.

Main Contributions.

- We introduce Reward-Aligned Distillation (RAD), a bilevel framework that assigns bounded, signed token weights according to their predicted effect on task reward (Section 4.2). A one-step approximation yields optimizer-aware scores computed jointly through a single Jacobian–vector product (Section 4.3).
- At matched training-response budgets, RAD improves mean accuracy across six math benchmarks over the strongest evaluated baselines by **6.5** and **2.5** percentage points for Qwen3-1.7B (non-thinking) and Qwen3-4B-Instruct-2507, respectively. On LiveCodeBench-v6, the corresponding pass@1 gains are **8.7** and **6.4** points for Qwen3-1.7B and OLMo3-7B-Instruct (Section 5).
- We demonstrate compatibility with an alternative distillation signal by applying RAD to OPD[2]. This improves four-benchmark mean math accuracy by **2.66** and **6.51** percentage points for the 1.7B and 4B Qwen students, respectively (Section 5.6).

2 RELATED WORK

On-policy distillation for LLMs. On-policy distillation queries a teacher on student-generated responses, reducing the training–inference mismatch of offline distillation (Agarwal et al., 2024; Gu et al., 2024). Related self-distillation methods obtain teacher signals from the student itself using privileged information such as verified solutions or environment feedback (Shenfeld et al., 2026; Hübotter et al., 2026; Zhao et al., 2026).

Unifying distillation and RLVR. KDRL (Xu et al., 2025) & RLAD (Zhang et al., 2026) combine verifiable rewards with teacher guidance, while RG-OPD (Akhondzadeh et al., 2026) uses verifier outcomes and teacher likelihoods to gate trajectory-level distillation. ExOPD (Yang et al., 2026) scales implicit teacher-derived rewards relative to KL regularization.

Token-selective distillation. TIP (Xu et al., 2026) selects tokens using uncertainty and teacher–student divergence, while DEAR (Xiao et al., 2026) identifies decision tokens and supporting evidence. REOPOLD (Ko et al., 2026) combines teacher-derived reward clipping with entropy-guided token sampling. Armandpour et al. (2026) analyze reward–distillation gradient alignment and study oracle filtering of positively aligned tokens. RAD estimates reward sensitivity through the shared student parameters and optimizer to assign bounded, signed weights during training. Signed weights additionally permit reversing teacher contributions; restricting RAD to nonnegative weights reduces accuracy in our Qwen3-1.7B math ablation (Table 2). We compare against representative baselines from both reward-aware and token-selective distillation literature. Extended related work appears in Appendix B.

3 BACKGROUND

Let $\boldsymbol{x} \sim \mathcal{D}$ denote a prompt sampled from the data distribution $\mathcal{D}$. We condition on $\boldsymbol{x}$ and omit its outer expectation throughout for simplicity. Let $\pi_{\boldsymbol{\theta}}$ denote the student model, $\boldsymbol{y} = (y_1, \ldots, y_T) \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})$ be a response sampled from the student, and $s_t = (\boldsymbol{x}, y_{<t})$ be the autoregressive state.

3.1 REINFORCEMENT LEARNING WITH VERIFIABLE REWARDS

Reinforcement learning with verifiable rewards (RLVR) trains a policy to maximize rewards from verifiable outcomes, such as correct answers or passing code tests. Given a reward function $R(\boldsymbol{x}, \boldsymbol{y})$, the expected-reward objective is defined as:

$$J_R(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \left[R(\boldsymbol{x}, \boldsymbol{y}) \right].$$

Several policy-gradient surrogates can be used to optimize this objective in RLVR, including proximal policy optimization (PPO) (Schulman et al., 2017) and group relative policy optimization (GRPO) (Shao et al., 2024). In this work, we use GRPO's group-relative advantages to estimate the reward gradient. For each prompt, we sample G responses $\{\boldsymbol{y}_i\}_{i=1}^{G}$ from the student and normalize their rewards within the group: $A_i^R = \frac{R(\boldsymbol{x}, \boldsymbol{y}_i) - \mu_R}{\sigma_R}, \mu_R = \frac{1}{G} \sum_{j=1}^{G} R(\boldsymbol{x}, \boldsymbol{y}_j)$, where σ_R is the group reward standard deviation. Every token in $\boldsymbol{y}_i$ receives the same response-level advantage A_i^R.

3.2 ON-POLICY DISTILLATION

On-policy distillation (OPD) samples responses from the student and evaluates the teacher on the same student-generated states (Agarwal et al., 2024; Gu et al., 2024). For a fixed teacher π_{T}, the reverse-KL objective is defined as:

$$L_D(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \left[\sum_{t=1}^{T} \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \right].$$

Unlike GRPO's sequence-level reward, OPD provides dense supervision at every generated token. Standard OPD, however, weights all token-level teacher signals equally. Moreover, minimizing L_D alone provides no monotonic improvement guarantee for J_R.

4 REWARD-ALIGNED DISTILLATION

Reward-Aligned Distillation (RAD) selects token-level distillation weights according to their predicted effect on task reward. We introduce the weighted objective (§4.1), then develop a bilevel formulation and its one-step approximation (§4.2). We extend the resulting scores to general optimizers and compute them efficiently using Jacobian–vector products (§4.3). Algorithm 1 summarizes our training recipe. Finally, we present RAD-Lite, a simplified variant that reduces computational overhead (§4.5).

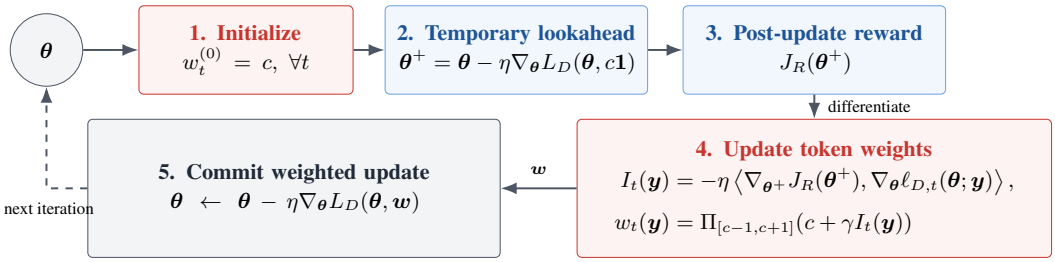

Figure 2: **One iteration of reward-aligned distillation (RAD), illustrated with SGD.** (1) Starting from student $\boldsymbol{\theta}$, assign the initial token weight c to every token. (2) Take one temporary step on the weighted distillation objective to obtain $\boldsymbol{\theta}^+$, as in Equation (3). (3) Sample separate meta rollouts from $\boldsymbol{\theta}^+$ to estimate the post-update reward gradient. (4) Differentiate this reward through the temporary step to score each token using Equation (4). Positive scores increase token weights and negative scores decrease them; projected ascent clips the updated weights to $[c-1, c+1]$, as in Equation (5). (5) Discard the temporary update and take one weighted distillation step from the original $\boldsymbol{\theta}$. Reset the weights to c for the next batch.

4.1 Token-Weighted Distillation

Let π_{old} denote the sampling policy. Holding sampled prefixes fixed, we define the per-token distillation surrogate

$$\ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) = \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\text{old}}(y_t \mid s_t)} \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\text{T}}(y_t \mid s_t)}.$$

Assigning real-valued token weights $\boldsymbol{w}(\boldsymbol{y}) = (w_1(\boldsymbol{y}), \ldots, w_T(\boldsymbol{y}))$, with prompt dependence suppressed, gives the weighted objective

$$L_D(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\text{old}}(\cdot \mid \boldsymbol{x})} \left[\sum_{t=1}^{T} w_t(\boldsymbol{y}) \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) \right]. \tag{1}$$

The weights can strengthen, suppress, or reverse individual distillation contributions. Uniform weights $w_t = 1$ recover the fixed-prefix OPD surrogate. Appendix F derives the corresponding gradient.

4.2 Bilevel Token Weighting

Recall that our goal is to choose token weights in order to maximize the reward after distillation. This naturally leads to a bilevel optimization formulation:

$$\max_{\boldsymbol{w}} \ J_R(\boldsymbol{\theta}^\star(\boldsymbol{w})) \quad \text{s.t.} \quad \boldsymbol{\theta}^\star(\boldsymbol{w}) \in \arg\min_{\boldsymbol{\theta}} L_D(\boldsymbol{\theta}, \boldsymbol{w}). \tag{2}$$

The inner problem trains the student with weighted distillation, and the outer problem chooses weights to maximize the trained student's reward. Solving the inner problem for each weight update is computationally infeasible. Therefore, we approximate its solution with one SGD (Robbins & Monro, 1951) update from the current student, with learning rate $\eta > 0$:

$$\boldsymbol{\theta}^\star(\boldsymbol{w}) \approx \boldsymbol{\theta}^+(\boldsymbol{w}) = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L_D(\boldsymbol{\theta}, \boldsymbol{w}). \tag{3}$$

At each iteration, we set the initial token weights $w_t^{(0)} = c$ for every token and denote the temporary student $\boldsymbol{\theta}^+ = \boldsymbol{\theta}^+(c\mathbf{1})$. For the sampled objective, differentiating the post-update reward through Equation (3) with respect to each token's weight gives the token score, omitting the loss normalization:

$$\left. \frac{\partial J_R(\boldsymbol{\theta}^+(\boldsymbol{w}))}{\partial w_t(\boldsymbol{y})} \right|_{\boldsymbol{w}=c\mathbf{1}} = -\eta \left\langle \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+), \nabla_{\boldsymbol{\theta}} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) \right\rangle =: I_t(\boldsymbol{y}). \tag{4}$$

A positive token score favors increasing the token's weight, while a negative score favors decreasing it. We update the weights by one projected ascent step:

$$w_t(\boldsymbol{y}) = \Pi_{[c-1,c+1]}(c + \gamma I_t(\boldsymbol{y})), \qquad \gamma > 0. \tag{5}$$

Π clips to the weights to be inside $[c-1, c+1]$ to prevent extreme outlier values and γ is the effective weight step size. We will discuss the choice of initial weight c in the next section. We reset the weights to c for each new iteration, discard the temporary update, and train the original student with the updated weights. Figure 2 summarizes the iteration.

Algorithm 1 Reward-aligned distillation (RAD)

Input: Student $\pi_{\boldsymbol{\theta}}$, teacher π_{T}, reward R, distillation coefficient λ, optimizer step $\mathcal{A}$, initial token weight c, effective weight step size γ.

for each training iteration **do**
  Sample $\mathcal{B} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_i$ from $\pi_{\boldsymbol{\theta}}$ and score its tokens with the teacher.
  Define weighted distillation loss $\widehat{L}_D(\boldsymbol{\theta}, \boldsymbol{w}; \mathcal{B}) = \frac{\lambda}{N} \sum_{i,t} w_{i,t} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}_i)$, where N is the number of valid response tokens in $\mathcal{B}$.
  Initialize $\boldsymbol{w}^{(0)} \leftarrow c\mathbf{1}$; save parameters and optimizer state.
  Take the temporary step $\boldsymbol{\theta}^+ \leftarrow \mathcal{A}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \widehat{L}_D(\boldsymbol{\theta}, c\mathbf{1}; \mathcal{B}))$.
  Sample a separate meta batch from the training dataset, generate meta rollouts with $\pi_{\boldsymbol{\theta}^+}$, and estimate $\nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+)$.
  Compute $\boldsymbol{p}$ using Equation (6).
  Restore the original parameters and optimizer state.
  Compute $\mathbf{I} \leftarrow -\operatorname{JVP}_{\boldsymbol{\theta}}[\boldsymbol{\ell}_D(\boldsymbol{\theta}; \mathcal{B})][\boldsymbol{p}]$.
  Update $w_{i,t} \leftarrow \Pi_{[c-1,c+1]}(c + \gamma I_{i,t})$.
  Train the student $\boldsymbol{\theta} \leftarrow \mathcal{A}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \widehat{L}_D(\boldsymbol{\theta}, \boldsymbol{w}; \mathcal{B}))$, with detached $\boldsymbol{w}$.
end for

4.3 Optimizer-Aware Scoring via Jacobian–Vector Product (JVP)

We now extend the SGD derivation to general optimizers, including AdamW (Loshchilov & Hutter, 2019), which is widely used in LLM training. This extension accounts for how the student optimizer transforms the weighted distillation gradient. Let $\mathcal{A}(\boldsymbol{\theta}, \boldsymbol{g})$ denote the parameters returned by one optimizer step with gradient $\boldsymbol{g}$, treating the pre-step optimizer state as fixed. Define $\boldsymbol{g}(\boldsymbol{w}) = \nabla_{\boldsymbol{\theta}} L_D(\boldsymbol{\theta}, \boldsymbol{w})$. The one-step approximation becomes

$$\max_{\boldsymbol{w}} \; J_R\big(\boldsymbol{\theta}^+(\boldsymbol{w})\big) \quad \text{s.t.} \quad \boldsymbol{\theta}^+(\boldsymbol{w}) = \mathcal{A}(\boldsymbol{\theta}, \boldsymbol{g}(\boldsymbol{w})).$$

We evaluate the token-weight derivatives at $\boldsymbol{w} = c\mathbf{1}$, writing $\boldsymbol{g} = \boldsymbol{g}(c\mathbf{1})$ and $\boldsymbol{\theta}^+ = \boldsymbol{\theta}^+(c\mathbf{1})$. Similar to Equation (4), differentiating the post-update reward through this optimizer step gives the *optimizer-aware token score*,

$$I_t(\boldsymbol{y}) = -\left\langle \nabla_{\boldsymbol{\theta}} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}), \boldsymbol{p} \right\rangle, \qquad \boldsymbol{p} = -\left(\frac{\partial \mathcal{A}(\boldsymbol{\theta}, \boldsymbol{g})}{\partial \boldsymbol{g}} \right)^{\top} \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+). \tag{6}$$

For SGD, $\boldsymbol{p} = \eta \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+)$, recovering Equation (4). At $c = 0$, the distillation gradient vanishes, so plain SGD without momentum or weight decay gives $\boldsymbol{\theta}^+ = \boldsymbol{\theta}$. AdamW can still move through its existing moments and decoupled weight decay; token-weight derivatives can remain nonzero in either case. For AdamW, we differentiate through the moment updates and gradient clipping; the derivation is in Appendix G. Let $\boldsymbol{\ell}_D$ collect the per-token surrogates. We compute all scores together using one JVP:

$$\mathbf{I}(\boldsymbol{y}) = -\operatorname{JVP}_{\boldsymbol{\theta}}[\boldsymbol{\ell}_D(\boldsymbol{\theta}; \boldsymbol{y})][\boldsymbol{p}] \in \mathbb{R}^{|\boldsymbol{y}|}. \tag{7}$$

The AdamW-specialized JVP is given in Equation (13). We implement a custom Triton kernel for the flash attention JVP; Appendix H gives the derivation and pseudocode (Algorithm 2).

Choice of initial token weight. The initial token weight c determines both the temporary distillation update and the starting point for projected ascent. Under the symmetric constraint $[c-1, c+1]$, $c = 1$ starts from uniform distillation weights and learns a weight perturbation around 1 within $[0, 2]$, while $c = 0$ starts from zero weights and permits signed weights in $[-1, 1]$. We treat c as a hyperparameter that may vary between iterations. Appendix A.4 compares these initialization choices. Experiments indicate that $c = 0$ learns slower than $c = 1$ but can achieve better final performance. Gradually expanding the negative range from $[0, 1]$ to $[-1, 1]$ for $c = 0$ can help balance learning speed and performance.

4.4 RAD Training Algorithm

Distillation rollouts are used to update the student, whereas meta rollouts are used to estimate the reward gradient for token weighting. Both prompt batches are sampled from the same training dataset.

Table 1: Math accuracy (%): RAD at step 100 versus baselines at step 200. Values are mean ± standard deviation over 16 sample-position evaluations from one run; Mean equally weights six benchmarks. Bold/underline mark the best/second-best trained means, with ties sharing a rank. Parentheses give RAD's gains over the best baseline in percentage points.

Method	AIME24	AIME25	AIME26	HMMT25 Feb	HMMT25 Nov	Olympiad Bench	Mean
Teacher	$74.79_{\pm 3.72}$	$61.46_{\pm 3.53}$	$74.79_{\pm 4.56}$	$43.75_{\pm 5.38}$	$59.17_{\pm 5.59}$	$77.15_{\pm 0.52}$	$65.18_{\pm 1.66}$
Student: Qwen3-1.7B (non-thinking)							
Base	$12.08_{\pm 3.89}$	$11.04_{\pm 4.20}$	$8.13_{\pm 2.63}$	$6.04_{\pm 3.38}$	$3.75_{\pm 2.00}$	$39.30_{\pm 0.82}$	$13.39_{\pm 1.49}$
GRPO	$29.58_{\pm 5.64}$	$19.58_{\pm 3.89}$	$18.96_{\pm 5.98}$	$15.63_{\pm 2.82}$	$10.21_{\pm 4.93}$	$53.07_{\pm 0.93}$	$24.51_{\pm 1.46}$
OPD	$\underline{40.00_{\pm 5.65}}$	$27.29_{\pm 4.44}$	$26.46_{\pm 5.83}$	$\underline{17.71_{\pm 3.67}}$	$16.67_{\pm 6.24}$	$58.19_{\pm 1.17}$	$31.05_{\pm 2.01}$
TIP	$39.17_{\pm 4.93}$	$28.54_{\pm 4.40}$	$31.04_{\pm 5.98}$	$17.08_{\pm 3.09}$	$18.33_{\pm 4.08}$	$57.93_{\pm 1.18}$	$32.02_{\pm 1.70}$
RLAD	$35.21_{\pm 4.08}$	$27.71_{\pm 4.20}$	$26.04_{\pm 6.89}$	$16.88_{\pm 3.43}$	$17.08_{\pm 4.55}$	$58.06_{\pm 1.27}$	$30.16_{\pm 1.58}$
ExOPD	$36.88_{\pm 4.33}$	$\underline{31.67_{\pm 4.08}}$	$\underline{31.25_{\pm 5.51}}$	$16.67_{\pm 2.89}$	$\underline{19.38_{\pm 5.43}}$	$\underline{59.69_{\pm 1.12}}$	$\underline{32.59_{\pm 1.81}}$
RAD	$\mathbf{47.50_{\pm 7.41}}$	$\mathbf{36.04_{\pm 3.58}}$	$\mathbf{38.96_{\pm 3.86}}$	$\mathbf{22.50_{\pm 4.17}}$	$\mathbf{24.79_{\pm 4.08}}$	$\mathbf{64.60_{\pm 1.05}}$	$\mathbf{39.07_{\pm 1.59}}$
Δ vs. best baseline	(+7.50)	(+4.38)	(+7.71)	(+4.79)	(+5.42)	(+4.92)	(+6.48)
Student: Qwen3-4B-Instruct-2507							
Base	$64.17_{\pm 5.95}$	$48.33_{\pm 5.53}$	$53.13_{\pm 5.46}$	$32.29_{\pm 4.52}$	$44.58_{\pm 4.06}$	$71.77_{\pm 0.96}$	$52.38_{\pm 1.47}$
GRPO	$\underline{67.92_{\pm 6.44}}$	$\underline{55.63_{\pm 3.67}}$	$\mathbf{62.29_{\pm 4.52}}$	$\underline{34.58_{\pm 3.51}}$	$\underline{48.75_{\pm 5.38}}$	$\underline{74.58_{\pm 0.84}}$	$\underline{57.29_{\pm 1.72}}$
OPD	$65.21_{\pm 6.23}$	$50.63_{\pm 4.60}$	$\underline{55.42_{\pm 5.25}}$	$33.96_{\pm 3.95}$	$41.46_{\pm 5.00}$	$71.89_{\pm 0.74}$	$53.09_{\pm 1.72}$
TIP	$61.04_{\pm 4.82}$	$49.79_{\pm 5.71}$	$53.96_{\pm 6.79}$	$33.96_{\pm 4.44}$	$41.04_{\pm 6.09}$	$71.77_{\pm 0.83}$	$51.93_{\pm 1.85}$
RLAD	$59.79_{\pm 5.06}$	$47.71_{\pm 4.96}$	$53.33_{\pm 6.12}$	$32.29_{\pm 4.67}$	$39.58_{\pm 5.64}$	$72.50_{\pm 0.93}$	$50.87_{\pm 1.60}$
ExOPD	$58.13_{\pm 5.13}$	$50.42_{\pm 3.51}$	$55.21_{\pm 5.52}$	$33.33_{\pm 4.08}$	$40.63_{\pm 5.17}$	$71.26_{\pm 0.92}$	$51.49_{\pm 2.02}$
RAD	$\mathbf{71.25_{\pm 5.51}}$	$\mathbf{62.50_{\pm 3.63}}$	$\mathbf{62.29_{\pm 5.23}}$	$\mathbf{36.04_{\pm 5.30}}$	$\mathbf{51.25_{\pm 5.12}}$	$\mathbf{75.46_{\pm 0.87}}$	$\mathbf{59.80_{\pm 2.07}}$
Δ vs. best baseline	(+3.33)	(+6.88)	(+0.00)	(+1.46)	(+2.50)	(+0.88)	(+2.51)

RAD's bilevel formulation makes token weighting depend on the student's post-update reward, introducing additional computation before each weighted distillation update. At each step, we take a temporary AdamW step with initial token weights $c\mathbf{1}$, generate separate meta rollouts, and estimate the reward gradient using GRPO objective. We compute the optimizer-aware token scores via a JVP (Equation (7)), update the weights with Equation (5), and train the original student with the updated weights. The meta batch is only used to learn the token weights; the committed student update uses the distillation rollouts. Algorithm 1 summarizes the procedure of RAD, and Appendix A.2 gives the batch objective and implementation details. RAD can in principle combine any differentiable token-level OPD objective with any RLVR gradient estimator. Section 5.6 demonstrates compatibility with the OPD^2 (Heo et al., 2026) distillation signal.

4.5 *RAD-Lite*: A LIGHTWEIGHT APPROXIMATION OF RAD

The bilevel formulation adds a temporary optimizer update, meta rollouts, a reward-gradient computation, and a JVP to standard OPD. To reduce this cost, *RAD-Lite* estimates the pre-update reward gradient by reusing the distillation rollouts. Replacing the post-update reward gradient in Equation (4) by the pre-update reward gradient gives

$$I_t(\boldsymbol{y}) = -\eta \left\langle \nabla_{\boldsymbol{\theta}} J_R(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) \right\rangle .$$

We use the same projected weight update in Equation (5). RAD-Lite also omits the optimizer Jacobian, while the student update still uses AdamW. Reusing the distillation rollouts avoids the temporary student update and separate meta rollout generation, halving the number of training responses per step. The only additional computation compared with OPD are reward-gradient estimation and a JVP. RAD-Lite ignores changes in the reward gradient after distillation and the optimizer's effect on token scores. Section 5.5 compares RAD-Lite with RAD using similar training configuration.

5 EXPERIMENTS

We describe the experimental setup and evaluation protocol, then present results and ablations. Implementation details, including inference hyperparameters, appear in Appendix A.

5.1 EXPERIMENTAL SETTINGS

Models. For mathematical reasoning, we use Qwen3-30B-A3B-Instruct-2507 as the teacher and Qwen3-1.7B (non-thinking) and Qwen3-4B-Instruct-2507 as students (Yang et al., 2025). For code generation, we evaluate Qwen3-1.7B (non-thinking) with the same teacher, and OLMo3-7B-Instruct with OLMo3.1-32B-Instruct as its teacher.

Training protocol. We train with `verl` (Sheng et al., 2025) using 64 prompts and eight responses per prompt. Within each domain and student setting, all methods share the training data, student initialization, rollout group size, sampling configuration, and optimizer settings. RAD additionally samples an equally sized, separate meta batch from the temporary student and updates token weights using exact JVPs. We compare 100 RAD updates with 200 baseline updates, matching 102,400 generated training responses, including RAD's meta rollouts. At this budget, RAD uses 17–18% more training time than OPD and RLAD on Qwen3-1.7B math, but less than TIP and ExOPD. On OLMo3-7B coding, RAD uses 18–39% less training time than the reported baselines (Table 11). RAD uses the initial token weight $c = 0$ in the main math and coding comparisons, with an additional cosine schedule of negative range for coding. Shared and method-specific settings appear in Appendix A.1.

Baselines. We compare RAD with GRPO (Shao et al., 2024), OPD (Agarwal et al., 2024), TIP (Xu et al., 2026), RLAD (Zhang et al., 2026), and ExOPD (Yang et al., 2026). Given the higher training cost of the OLMo3-7B setting (Table 11), we retain GRPO and OPD as reference methods, with TIP and RLAD representing strongest token-selective and reward-aware distillation baselines, respectively. Baseline-specific hyperparameters follow the respective papers.

Mathematical reasoning. Following ExOPD (Yang et al., 2026), we train on 57,046 problems from DeepMath-103K (He et al., 2025) with difficulty ratings of at least 6. We evaluate on AIME24 (AI-MO, 2024), AIME25 (OpenCompass, 2025), AIME26, HMMT2025-Feb, HMMT2025-Nov (Balunovic et al., 2025), and OlympiadBench (He et al., 2024). We sample 16 responses per problem at temperature 1.0 and top-p 1.0, with a 32,768-token generation limit. Responses are graded with Math-Verify [1], and pass@1 is estimated as the fraction of sampled responses that are correct. The overall score is the unweighted mean across the six benchmarks.

Code generation. We train on the Eurus code split (Cui et al., 2025), assigning reward 1 when all tests pass and 0 otherwise. We evaluate on all 1,055 LiveCodeBench-v6 tasks (Jain et al., 2025a). Following ExOPD (Yang et al., 2026), we sample four responses per task at temperature 1.0 and top-p 1.0, with a 32,768-token generation limit. Dataset statistics appear in Table 5.

5.2 MATHEMATICAL REASONING RESULTS

At the matched training-response budget (Table 1), RAD reaches **39.07%** mean accuracy with Qwen3-1.7B (non-thinking), exceeding the strongest evaluated baseline, ExOPD at 32.59%, by **6.48** points. With Qwen3-4B-Instruct-2507, RAD reaches **59.80%**, exceeding the strongest evaluated baseline, GRPO at 57.29%, by **2.51** points. The corresponding gains over OPD are **8.01** and **6.71 points**.

For Qwen3-4B-Instruct-2507, OPD reaches 53.09% at step 200, slightly above the base student's 52.38%, while TIP, RLAD, and ExOPD perform below the base student. Similar degradation of OPD and ExOPD is observed in Heo et al. (2026) (Table 4 and 5). RAD's gain in this setting supports using post-update reward to guide the contribution of each teacher signal even when the initial teacher–student gap is small.

Figure 5 in Appendix A.3 tracks both students on four benchmarks (AIME24/25 and HMMT2025-Feb/Nov). Figure 10 in Appendix E.4 compares RAD at step 90 with GRPO at step 450 on math and coding, with GRPO using **2.5**× as many generated training responses. Appendix E.3 additionally compares RAD with baselines with a matched training time. Appendix E compares total training

[1] https://github.com/huggingface/Math-Verify

Table 2: Qwen3-1.7B (non-thinking) math ablations at step 100 ($c = 0$): mean accuracy over the four benchmarks (AIME24/25 and HMMT2025-Feb/Nov). Variants use separate meta rollouts unless rollout reuse is specified. Δ (pp) denotes difference from RAD. **Bold**/underline: best/second-best displayed means.

Variant	Change from RAD	Acc. (%)	Δ (pp)
RAD	AdamW lookahead; separate meta rollouts	<u>32.71</u>	—
Negative score	Reverse the score: $I_t \mapsto -I_t$	0.36	(-32.34)
Shuffled weights	Permute weights within each response	6.35	(-26.35)
Positive only	Restrict weights to $[0, 1]$	25.00	(-7.71)
Negative-bound warmup	Cosine lower bound: $0 \to -1$ over 50 steps	31.20	(-1.51)
Pre-update gradient	Reward gradient before the temporary update	**34.32**	$(+1.61)$
+ rollout reuse	Pre-update reward grad; reuse distillation rollouts	18.54	(-14.17)
SGD lookahead	SGD lookahead update	16.88	(-15.83)

times for Qwen3-1.7B (non-thinking) on math and OLMo3-7B-Instruct on coding, and reports a RAD training-time breakdown for Qwen3-1.7B (non-thinking).

5.3 Code Generation Results

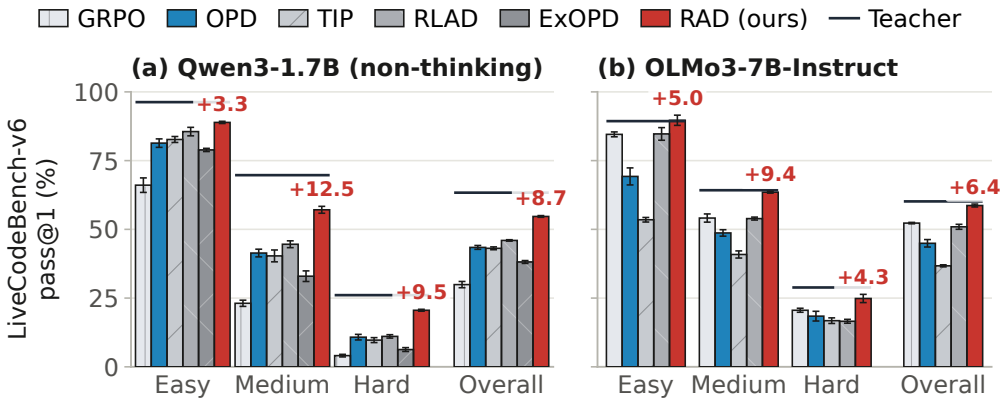

Figure 3: LiveCodeBench-v6 pass@1 (%) by difficulty, comparing RAD at step 100 with baselines at step 200. Numbers above RAD's bars give its improvement over the best baseline in that tier (percentage points). Error bars show ± 1 standard deviation over four sample positions; black lines mark available teacher means. Unavailable baseline results are omitted. Table 8 in Appendix C.1 reports the full means and standard deviations.

Figure 3 compares both students across varying difficulty tiers on full LiveCodeBench-v6. Appendix C.1 reports the full numerical results (Table 8). RAD with Qwen3-1.7B (non-thinking) reaches $\mathbf{54.69 \pm 0.31}\%$ pass@1 at step 100, compared with $45.97 \pm 0.26\%$ for RLAD, the strongest step-200 baseline. RAD also leads the five evaluated baselines on Easy, Medium, and Hard. Appendix C.2 reports Qwen3-1.7B (non-thinking) coding-validation dynamics and checkpoint results.

With OLMo3-7B-Instruct, RAD reaches $\mathbf{58.67 \pm 0.52}\%$ on full LiveCodeBench-v6 at step 100, exceeding GRPO at $52.27 \pm 0.26\%$, the strongest completed step-200 baseline, by 6.40 points. These results extend the coding comparison to a different model family and a larger student. Appendix C.3 reports OLMo3-7B-Instruct coding-validation dynamics and checkpoint results.

5.4 RQ1: Which components contribute to RAD's performance?

We ablate RAD using Qwen3-1.7B (non-thinking) with $c = 0$ after 100 updates on AIME24/25 and HMMT2025-Feb/Nov (Table 2). Reversing scores or shuffling weights reduces accuracy from 32.71%

to 0.36% and 6.35%, respectively, supporting token-specific alignment. Restricting weights to $[0, 1]$ yields 25.00%, while gradually introducing negative weights achieves 31.20%. Replacing AdamW lookahead with SGD yields 16.88%. These comparisons support signed weights and optimizer-aware scoring in this setting.

Evaluating the reward gradient before the temporary update achieves 34.32%, comparable to RAD, because the temporary update is negligible for initial token weights $c = 0$. Reusing distillation rollouts further reduces accuracy to 18.54%, but also halves the generated responses per update. These findings motivate evaluating simplified scoring at a matched response budget, as studied next.

5.5 RQ2: How effective is RAD-Lite?

We evaluate RAD-Lite (Section 4.5) on four math benchmarks using Qwen3-1.7B (non-thinking) and Qwen3-4B-Instruct-2507 (Table 3). We compare RAD at 100 updates with RAD-Lite at 200 updates, matching 102,400 generated training responses. On Qwen3-1.7B, RAD-Lite achieves 37.50% accuracy versus RAD's 32.71%, a 4.79-point improvement. However, on Qwen3-4B-Instruct-2507, RAD-Lite reaches only 37.29%, compared with RAD's 55.26%. Thus, the benefits of simplification are not consistent across students. Understanding when additional updates compensate for simplified token scoring, and whether this trade-off generalizes across models and tasks, remains an important direction for future work.

Table 3: RAD at step 100 and RAD-Lite at step 200. Mean math accuracy (%) over AIME24/25 and HMMT2025-Feb/Nov.

Method	Accuracy (%)
Qwen3-1.7B (non-thinking)	
RAD	$32.71_{\pm 2.23}$
RAD-Lite	$37.50_{\pm 2.48}$
Qwen3-4B-Instruct-2507	
RAD	$55.26_{\pm 2.46}$
RAD-Lite	$37.29_{\pm 1.73}$

5.6 RQ3: Is RAD compatible with other distillation signals?

We apply RAD's token weighting to our local TopK64 implementation of OPD^2 (Heo et al., 2026), which derives supervision from an instruction-tuned teacher and its paired base model. We evaluate on the same four math benchmarks; implementation details appear in Appendix A.1.
Table 4 compares OPD^2 + RAD after 100 updates with uniform OPD and OPD^2 after 200 updates, matching the number of generated training responses. Adding RAD improves OPD^2, supporting its compatibility with an alternative distillation signal. For Qwen3-4B-Instruct-2507, the combined method also approaches teacher accuracy (Figure 4, Appendix A).

Table 4: RAD compatibility with OPD^2: RAD at step 100 versus baselines at step 200. Values are mean math accuracy (%) over AIME24/25 and HMMT2025-Feb/Nov.

Method	Qwen3-1.7B (non-thinking)	Qwen3-4B-Instruct-2507
Teacher	59.79	59.79
Base	8.23	47.34
Uniform OPD	25.42	47.81
OPD^2	<u>28.33</u>	<u>53.13</u>
OPD^2 + RAD	**30.99**	**59.64**
Δ vs. OPD^2	+2.66	+6.51

6 Conclusion and Future Work

RAD uses optimizer-aware lookahead scores and projected ascent to align token-level distillation with reward. It improves math and coding accuracy at a matched training-response budget. Results with OLMo3-7B-Instruct extend the evidence to a different model family and a larger student, while OPD^2 results support compatibility with other distillation signals.

Limitations and future work. RAD requires a temporary student update, fresh outer rollouts, and an exact JVP, increasing its per-step cost relative to standard OPD. Developing cheaper score approximations or reusing outer-rollout computation is therefore an important direction. Future work should explore longer training horizons, and learned or non-verifiable rewards.

AI USE STATEMENT

In this work, we used generative AI tools for writing assistance (grammar, phrasing, and LaTeX formatting) and coding assistance (debugging and boilerplate for training/evaluation scripts). We have not used generative AI tools for generating research ideas, designing the method, or writing the technical content, claims, or analysis, and generating or annotating research data is not applicable to this work. Additionally, we used generative AI tools for literature search and summarization of related work. Note that the large language models studied in our experiments (the Qwen3-1.7B, Qwen3-4B-Instruct-2507, Qwen3.5-4B, and OLMo3-7B-Instruct students and the Qwen3-30B-A3B-Instruct-2507, Qwen3.6-27B, and OLMo3.1-32B-Instruct teachers) are objects of study rather than tools used to produce this paper.

We have reviewed all AI-assisted work. All AI-assisted text was read and edited by the authors; AI-assisted code was reviewed, tested, and verified for correctness by at least one author before being used in any reported experiment; and every reference surfaced by AI tools was checked against the original source. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies reward-aligned token weighting for on-policy distillation, using publicly available models, data, and benchmarks for mathematical reasoning and code generation; it involves no human subjects or personal data. Distilled students may inherit the teacher's errors and biases, and reward-driven weighting adds the usual risks of reward optimization. We evaluate only task accuracy and make no claims about the safety-relevant behavior of the distilled students.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure the reproducibility of our results. The full algorithm is described in Section 4 with pseudocode in Algorithm 1 and implementation details in Appendix A.2. All hyperparameters, model configurations, and training schedules are listed in Appendix A.1, and details of datasets, environments, preprocessing, and evaluation protocols are given in Section 5.1 and Appendix A. Hardware and compute requirements are reported in Appendix E. Each method is trained once; we report means and standard deviations over repeated evaluation samples (16 responses per problem for math, four for coding, and three rollouts per task for SWE-bench Verified). The token scores are derived in Sections 4.2 and 4.3, with complete derivations in Appendices F and G. We will release code and checkpoints upon acceptance.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

AI-MO. Aime 2024, 2024. URL https://huggingface.co/datasets/AI-MO/aimo-validation-aime.

Mohammad Sadegh Akhondzadeh, Vijay Lingam, Atula Tejaswi, Chanakya Ekbote, Sujay Sanghavi, and Aleksandar Bojchevski. Reward-gated on-policy distillation, 2026. URL https://arxiv.org/abs/2607.04037.

Mohammadreza Armandpour, Fatih Ilhan, David Harrison, Ajay Jaiswal, Duc NM Hoang, Fartash Faghri, Yizhe Zhang, Minsik Cho, and Mehrdad Farajtabar. Unmasking on-policy distillation: Where it helps, where it hurts, and why. *arXiv preprint arXiv:2605.10889*, 2026.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2025.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Neil Chowdhury et al. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Codeforces. Codeforces. URL https://codeforces.com/.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Chanakya Ekbote, Vijay Lingam, Behrooz Omidvar Tehrani, Jun Huan, Sujay Sanghavi, Anoop Deoras, and Stefano Soatto. Murphy: Reflective multi-turn reinforcement learning for self-correcting code generation in large language models. In *First Workshop on Foundations of Reasoning in Language Models*, 2025. URL https://openreview.net/forum?id=x0Ir7cWEiA.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, 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.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. DeepMath-103K: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Byeongho Heo, Jaehui Hwang, Sangdoo Yun, and Dongyoon Han. On-policy delta distillation. *arXiv preprint arXiv:2607.15161*, 2026.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, 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 *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025a.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025b.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Jongwoo Ko, Sara Abdali, Young Jin Kim, Tianyi Chen, and Pashmina Cameron. Scaling reasoning efficiently via relaxed on-policy distillation, 2026. URL https://arxiv.org/abs/2603.11137.

Jeongyeol Kwon, Dohyun Kwon, Stephen Wright, and Robert D Nowak. A fully first-order method for stochastic bilevel optimization. In *International Conference on Machine Learning*, pp. 18083–18113. PMLR, 2023.

Minchong Li, Feng Zhou, and Xiaohui Song. BiLD: Bi-directional logits difference loss for large language model distillation. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 1168–1182, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.78/.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. Bome! bilevel optimization made easy: A simple first-order approach. *Advances in neural information processing systems*, 35: 17248–17262, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Thang Luong and Edward Lockhart. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad, 2025. URL https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/.

OpenCompass. Aime 2025, 2025. URL https://huggingface.co/datasets/opencompass/AIME2025.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pp. 4334–4343. PMLR, 2018.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. URL https://arxiv.org/abs/1910.01108.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

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*, pp. 1279–1297, 2025.

Jinwei Xiao, Zhuowen Han, Yueqing Sun, Zhengxi Lu, Yuxin Liu, Zhiyuan Yao, Wentao Chen, Qi Gu, and Xunliang Cai. Finding the evidence: Discovering decision-supporting tokens for on-policy reasoning distillation. *arXiv preprint arXiv:2606.22830*, 2026.

Hongling Xu, Qi Zhu, Heyuan Deng, Jinpeng Li, Lu Hou, Yasheng Wang, Lifeng Shang, Ruifeng Xu, and Fei Mi. Kdrl: Post-training reasoning llms via unified knowledge distillation and reinforcement learning. *arXiv preprint arXiv:2506.02208*, 2025.

Yuanda Xu, Hejian Sang, Zhengze Zhou, Ran He, Zhipeng Wang, and Alborz Geramifard. Tip: Token importance in on-policy distillation. *arXiv preprint arXiv:2604.14084*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*, 2026.

Zhaoyang Zhang, Shuli Jiang, Yantao Shen, Yuting Zhang, Dhananjay Ram, Shuo Yang, Zhuowen Tu, Wei Xia, and Stefano Soatto. Reinforcement-aware knowledge distillation for llm reasoning. *arXiv preprint arXiv:2602.22495*, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Appendices

CONTENTS

1 Introduction **2**

2 Related Work **2**

3 Background **3**

3.1 Reinforcement Learning with Verifiable Rewards 3

3.2 On-Policy Distillation 3

4 Reward-Aligned Distillation **3**

4.1 Token-Weighted Distillation 4

4.2 Bilevel Token Weighting 4

4.3 Optimizer-Aware Scoring via Jacobian–Vector Product (JVP) 5

4.4 RAD Training Algorithm 5

4.5 *RAD-Lite*: a lightweight approximation of RAD 6

5 Experiments **7**

5.1 Experimental Settings 7

5.2 Mathematical Reasoning Results 7

5.3 Code Generation Results 8

5.4 RQ1: Which components contribute to RAD's performance? 8

5.5 RQ2: How effective is RAD-Lite? 9

5.6 RQ3: Is RAD compatible with other distillation signals? 9

6 Conclusion and Future Work **9**

A Experimental Details and Additional Math Results **17**

A.1 Training Settings 17

A.2 Implementation Details 17

A.3 Math Evaluation Dynamics 19

A.4 Initial Token Weight Ablation 19

B Extended Related Work **21**

C Additional Coding Results **21**

C.1 Detailed LiveCodeBench Results 21

C.2 Qwen Coding Validation 21

C.3 OLMo3-7B Settings and Validation 23

D Preliminary Experiments on Agentic RL Settings **24**

E Computational Cost and Efficiency **24**

E.1 Comparison with Baselines 24

E.2 RAD Stage Breakdown 25

E.3 Math Comparison at Comparable Training Times 26

E.4 Extended GRPO Training 27

F Gradient of Weighted On-Policy Distillation **27**

G Hypergradient through One Optimizer Step **29**

H Custom Triton Kernel for Attention JVPs **30**

H.1 Layer-Streamed JVP Execution 30

A Experimental Details and Additional Math Results

A.1 Training Settings

Baseline implementations. GRPO optimizes task reward without teacher guidance, while OPD applies reverse-KL distillation uniformly to all response tokens. TIP uses Soft-OR selection with 50% token retention. RLAD modifies the clipped GRPO update with a teacher-aware policy ratio. ExOPD constructs a reward-extrapolated reverse-KL advantage using frozen student and teacher base policies. OPD, RLAD, ExOPD, and RAD use sampled-action estimates of reverse KL, while TIP uses full-vocabulary reverse KL.

Tables 6 and 7 record the main math settings. Let N_1 be the total number of valid response tokens in the first actor batch of each run, and N the corresponding count in the current batch. RAD uses a weight step size of $10^5 N_1$, applied to the weight derivative in Equation (11), gives the effective $\gamma = 10^5 \lambda N_1 / N$ in Equation (5). The unprojected weight change is γI_t: the score I_t already includes the optimizer learning rate through the optimizer Jacobian and the scaling of the estimated outer reward gradient. The weight step size should therefore be interpreted together with this score scale; the resulting weights are clipped to the specified interval. We apply no further weight normalization. Both math runs use the fixed initial token weight $c = 0$ and clipping interval $[-1, 1]$. These are experimental settings, while Equation (5) keeps c configurable.

Coding datasets and evaluation. The Eurus code split is assembled from APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022), Codeforces (Codeforces), and TACO (Li et al., 2023). LiveCodeBench-v6 contains 322 easy, 383 medium, and 350 hard tasks. Evaluation uses a 40,960-token context limit for the combined prompt and response, with at most 32,768 generated tokens per response.

The Qwen3-1.7B (non-thinking) coding RAD run uses the same weight step size $10^5 N_1$, and distillation coefficient $\lambda = 0.1$, without further weight normalization. It keeps the teacher's thinking prefix, uses the initial token weight $c = 0$, and clips weights to an upper bound of 1 and a lower bound that follows a cosine schedule from 0 to -1 over the first 50 steps. The initial token weight stays fixed throughout this clipping-bound schedule, which gradually permits negative weights. The OLMo coding run uses the same initial token weight and clipping schedule (Appendix C.3). The main coding comparison reports RAD at step 100 and baselines at step 200.

For the OPD^2 compatibility experiment, the teacher pair is Qwen3-30B-A3B-Instruct-2507 and Qwen3-30B-A3B-Base. Our local implementation uses frozen-student TopK64 support and global token-mean reduction, with distillation coefficient $\lambda = 0.1$ for both OPD^2 and OPD^2 + RAD. The RAD variant uses a cosine schedule for the initial token weight c from 1 to 0, a clipping radius of 1 around c, and a weight step size of $10^5 N_1$ with inverse length scaling. This schedule changes both the initial token weight and the clipping interval, moving the latter from $[0, 2]$ to $[-1, 1]$. Each step uses 64 actor prompts with eight responses each; RAD additionally uses 64 meta prompts with eight responses each. We compare OPD^2 + RAD at step 100 with uniform OPD and OPD^2 at step 200.

All experiments use eight NVIDIA H100 GPUs. Dataset statistics and metrics are reported in Table 5.

For AIME26, we use the 30-problem MathArena snapshot with two upstream-confirmed transcription corrections. For OlympiadBench, we use the 675-problem Qwen2.5-Math evaluation snapshot.

A.2 Implementation Details

Batch objective. Let $\mathcal{B} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_i$ be the current student rollout batch, with prefixes $s_{i,t} = (\boldsymbol{x}_i, y_{i,<t})$, and let N be its number of valid response tokens. The temporary and committed updates use the normalized batch objective

$$\widehat{L}_D(\boldsymbol{\theta}, \boldsymbol{w}; \mathcal{B}) = \frac{\lambda}{N} \sum_{i,t} w_{i,t} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}_i), \tag{8}$$

where λ is the distillation coefficient. We keep the loss normalization fixed when differentiating with respect to the weights. The common factor λ/N in their reward gradient is absorbed into the effective step size γ in Equation (5). We also apply a detached token-level rollout/training importance

Table 5: Dataset statistics and evaluation metrics. Counts denote unique problems or tasks; responses per prompt are sampled whenever a prompt is selected. RAD additionally samples an independent outer batch of 64 meta prompts with eight responses each (512 responses). The coding-validation rows form the 1,024-problem Eurus validation split.

Domain	Use	Dataset	# Problems	Responses / prompt	Metric
Math	Train	DeepMath-103K, level ≥ 6	57,046	8	Binary correctness reward
Math	Eval	AIME24	30	16	pass@1 accuracy (Math-Verify)
Math	Eval	AIME25	30	16	pass@1 accuracy (Math-Verify)
Math	Eval	AIME26	30	16	pass@1 accuracy (Math-Verify)
Math	Eval	HMMT2025-Feb	30	16	pass@1 accuracy (Math-Verify)
Math	Eval	HMMT2025-Nov	30	16	pass@1 accuracy (Math-Verify)
Math	Eval	OlympiadBench	675	16	pass@1 accuracy (Math-Verify)
Code	Train	Eurus code split (APPS, CodeContests, Codeforces, and TACO)	25,276	8	Binary all-tests-pass reward
Code	Val	APPS	142	1	pass@1 functional correctness
Code	Val	CodeContests	377	1	pass@1 functional correctness
Code	Val	Codeforces	123	1	pass@1 functional correctness
Code	Val	TACO	382	1	pass@1 functional correctness
Code	Eval	LiveCodeBench-v6	1,055	4	pass@1 functional correctness (native execution)

correction clipped at 5, omitted from the notation. The weights are detached for the committed student update.

JVP backends. The `flash_triton` attention implementation combines a FlashAttention forward pass with a custom Triton kernel for the attention JVP (Algorithm 2). The Qwen3-1.7B (non-thinking) and Qwen3-4B-Instruct-2507 RAD experiments use this implementation to compute the model JVP on a temporary full student replica with its parameter direction. The `olmo3_layer_streamed` backend, used for OLMo3-7B-Instruct RAD coding, computes the model JVP one layer at a time: it gathers each parameter group and its direction, propagates activations and their tangents, and releases the gathered tensors after use. This execution strategy reduces parameter and direction storage while reusing the same `flash_triton` attention rule; Appendix H.1 gives the details.

Table 6: Shared math training hyperparameters.

Hyperparameter	Value
Training prompts per step	64
Rollouts per prompt	8
Actor epochs	1
Maximum prompt length	2,048
Maximum response length	32,768
Training temperature	1.0
Training top-p	1.0
Optimizer	AdamW
Learning rate	1×10^{-6}
Weight decay	0.01
Global gradient clipping	1.0
Random seed	42
RAD / baseline steps	100 / 200
Evaluation interval	10 steps
Responses per eval problem	16

Table 7: Method-specific math hyperparameters. N_1 is the valid response-token count in the first actor batch.

Hyperparameter	Value
Distillation coefficient (RAD / baselines)	0.1
RAD: meta prompts	64
RAD: initial token weight c	0
RAD: weight step size	$10^5 N_1$
TIP retention ratio	0.5
RLAD α	0.5
RLAD teacher log-ratio clamp	1.0
RLAD clipping thresholds	0.20 / 0.28
ExOPD λ	1.25

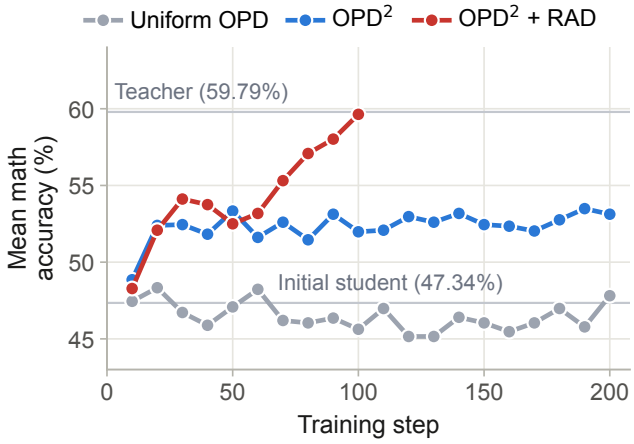

Figure 4: OPD^2 evaluation dynamics for Qwen3-4B-Instruct-2507, using Table 4's benchmarks and sampling. Horizontal lines mark the initial student and teacher.

A.3 Math Evaluation Dynamics

Figure 5 tracks mean accuracy on AIME24/25 and HMMT2025-Feb/Nov during training. For Qwen3-1.7B (non-thinking), the distillation baselines improve quickly and then plateau: ExOPD peaks at 28.65% at step 40 and reaches 26.15% at step 200. RAD improves more slowly early in training (17.19% at step 50), exceeds every baseline from step 70 onward, and reaches 32.71% at step 100. For Qwen3-4B-Instruct-2507, the distillation baselines remain between 44.22% and 48.70%, close to the base student (47.34%), and GRPO reaches 51.72% at step 200. RAD exceeds every baseline from step 30 onward and reaches 55.26% at step 100.

A.4 Initial Token Weight Ablation

How does the initial token weight c affect learning dynamics and final accuracy? In a four-run comparison, we study fixed $c = 0$, fixed $c = 1$, a cosine schedule from $c = 1$ to $c = 0$ over 100 updates, and fixed $c = 0$ with negative-weight warmup (Figure 6). The warmup keeps the upper clipping bound at 1 and lowers the lower bound from 0 to -1 with a cosine schedule through step 50, gradually permitting negative weights. All four runs use inverse length scaling without further weight normalization. At step 100, fixed $c = 0$ reaches 32.71% mean accuracy without warmup and

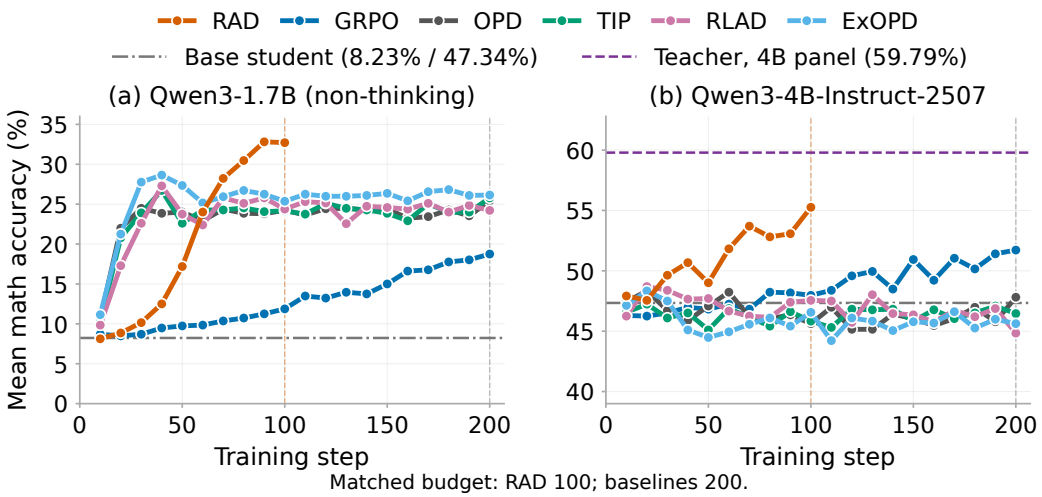

Figure 5: Math evaluation dynamics for Qwen3-1.7B (non-thinking) in the left panel and Qwen3-4B-Instruct-2507 in the right panel, averaged equally over AIME24/25 and HMMT2025-Feb/Nov with 16 sampled responses per problem. Horizontal lines mark base-student references and the teacher in the right panel.

31.20% with warmup, compared with 26.30% for fixed $c = 1$ and 27.45% for the cosine initialization schedule. Both zero-initialized variants outperform the other initialization choices at this checkpoint; fixed $c = 0$ without warmup leads the warmup variant by 1.51 percentage points in this single-seed comparison. The first three variants use $[c - 1, c + 1]$, varying initialization and clipping bounds together; warmup changes only the lower bound while keeping $c = 0$.

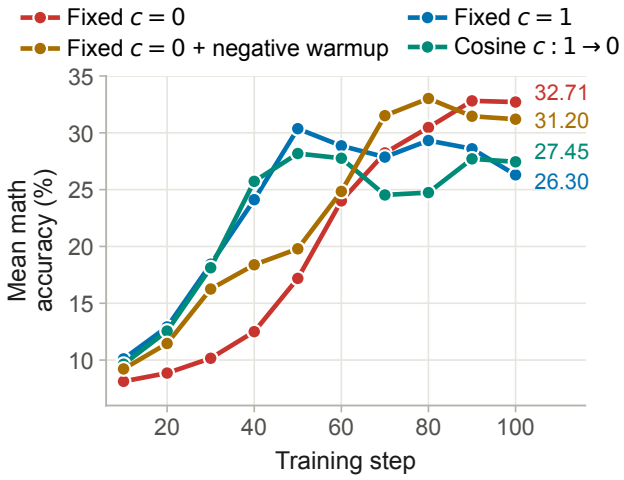

Figure 6: Initial token weight and negative-weight warmup comparison for Qwen3-1.7B (non-thinking). Each curve shows one run with seed 42 over the shared step-10–100 window. Accuracy equally averages AIME24/25 and HMMT2025-Feb/Nov, with 16 responses per problem. Fixed $c = 0$ without warmup uses the same run as Table 1, restricted to these four benchmarks. All runs use AdamW lookahead, separate meta rollouts, inverse length scaling, and no further weight normalization. The initialization variants clip to $[c - 1, c + 1]$; cosine initialization reaches $c = 0$ at step 100. Negative-weight warmup keeps $c = 0$ and the upper bound at 1 while lowering the lower bound from 0 to -1 by step 50 using a cosine schedule. Dots are observed evaluations; labels show step-100 accuracy.

B Extended Related Work

On-policy distillation for LLMs. Knowledge distillation transfers capabilities from a larger teacher to a smaller student (Buciluǎ et al., 2006; Hinton et al., 2015). For LLMs, teacher supervision can take the form of token-level probability distributions or generated text (Kim & Rush, 2016; Sanh et al., 2020). Off-policy distillation trains on fixed sequences, often generated by the teacher (Hsieh et al., 2023; Li et al., 2025). On-policy distillation instead queries the teacher on student-generated responses, exposing the student to supervision along its own trajectories and reducing the mismatch between training and inference (Agarwal et al., 2024; Gu et al., 2024). Self-distillation paradigm obtains supervision from the student itself, with privileged information such as verified solutions or environment feedback providing a stronger teacher context (Shenfeld et al., 2026; Hübotter et al., 2026; Zhao et al., 2026).

Unifying distillation and RLVR. Several methods combine dense teacher supervision with verifiable task rewards. KDRL (Xu et al., 2025) jointly optimizes a GRPO reward objective and reverse-KL distillation, while RLAD (Zhang et al., 2026) incorporates teacher guidance into policy optimization through a likelihood ratio anchored to a mixture of the teacher and the previous student policy. RG-OPD (Akhondzadeh et al., 2026) uses agreement between verifier outcomes and teacher likelihoods to gate trajectory-level distillation. A related perspective connects OPD to KL-regularized reinforcement learning: Yang et al. (2026) introduce a flexible reference model and scales the implicit reward derived from teacher probabilities, yielding the reward-extrapolation variant ExOPD. Our proposed method, RAD, uses task reward to determine how individual token-level distillation contributions should enter the student update.

Token-selective distillation. TIP (Xu et al., 2026) and DEAR (Xiao et al., 2026) select teacher supervision using uncertainty, teacher–student divergence, and reasoning structure. REOPOLD (Ko et al., 2026) combines token log-ratio rewards with clipping and entropy-guided sampling. Armandpour et al. (2026) diagnose token-level alignment between distillation and reward-improving directions in logit space and study oracle filtering of positively aligned tokens. RAD operationalizes reward-guided weighting during training: it differentiates a reward surrogate through the student optimizer and shared parameters to assign bounded, signed token weights. These weights can also reverse teacher contributions; our positive-only ablation supports the benefit of this additional flexibility. We compare RAD with representative reward-aware and token-selective baselines.

Bilevel reweighting. Learning example weights from a held-out objective is a standard use of bilevel optimization (Ren et al., 2018). Gradient similarity has also been used to adapt auxiliary losses toward a primary objective (Du et al., 2018). We apply this principle at token granularity in on-policy language-model distillation. The one-step approximation avoids solving the full inner problem or an implicit linear system (Liu et al., 2022; Kwon et al., 2023).

C Additional Coding Results

C.1 Detailed LiveCodeBench Results

Table 8 gives the full results underlying Figure 3, including standard deviations and gains for Qwen3-1.7B (non-thinking) and OLMo3-7B-Instruct.

C.2 Qwen Coding Validation

We evaluate Qwen3-1.7B (non-thinking), trained with Qwen3-30B-A3B-Instruct-2507 as teacher, on the full 1,024-problem Eurus validation split (Table 5). We sample one response per problem and report problem-weighted pass@1. Baselines generate 512 training responses per step; RAD generates 1,024 including meta rollouts. Training and weight settings appear in Appendix A.1.

At the matched training-response budget, RAD at step 100 reaches 29.69%, exceeding RLAD at step 200 (23.34%) by 6.35 points (Figure 7 and Table 9). Extended RAD training reaches 30.66% at step 200.

Table 8: Full LiveCodeBench-v6 pass@1 (%; mean ± standard deviation over four sample positions), comparing RAD at step 100 with baselines at step 200. Bold/underline: best/second-best trained means; parentheses: gains over the best evaluated baseline (pp, before rounding).

Method	Easy (322)	Medium (383)	Hard (350)	Overall (1,055)
Student: Qwen3-1.7B (non-thinking)				
Teacher	$96.27_{\pm 0.79}$	$69.71_{\pm 1.21}$	$26.07_{\pm 0.24}$	$63.34_{\pm 0.38}$
Base (student)	$60.02_{\pm 1.17}$	$16.45_{\pm 0.26}$	$1.79_{\pm 0.71}$	$24.88_{\pm 0.35}$
GRPO	$66.07_{\pm 2.64}$	$23.11_{\pm 1.13}$	$4.07_{\pm 0.47}$	$29.91_{\pm 1.14}$
OPD	$81.37_{\pm 1.54}$	$41.38_{\pm 1.40}$	$10.79_{\pm 1.02}$	$43.44_{\pm 0.70}$
TIP	$82.69_{\pm 1.08}$	$40.34_{\pm 2.16}$	$9.71_{\pm 0.90}$	$43.10_{\pm 0.52}$
RLAD	$\underline{85.56}_{\pm 1.55}$	$\underline{44.58}_{\pm 1.24}$	$\underline{11.07}_{\pm 0.62}$	$\underline{45.97}_{\pm 0.26}$
ExOPD	$78.88_{\pm 0.58}$	$32.96_{\pm 1.93}$	$6.29_{\pm 0.70}$	$38.13_{\pm 0.52}$
RAD (ours)	$\mathbf{88.90}_{\pm \mathbf{0.40}}$	$\mathbf{57.11}_{\pm \mathbf{1.26}}$	$\mathbf{20.57}_{\pm \mathbf{0.35}}$	$\mathbf{54.69}_{\pm \mathbf{0.31}}$
Δ vs. best baseline	(+3.34)	(+12.53)	(+9.50)	(+8.72)
Student: OLMo3-7B-Instruct				
Teacher	$89.36_{\pm 1.41}$	$64.23_{\pm 1.86}$	$28.86_{\pm 0.97}$	$60.17_{\pm 0.98}$
Base (student)	$76.09_{\pm 1.58}$	$43.41_{\pm 2.31}$	$14.86_{\pm 1.73}$	$43.91_{\pm 1.50}$
GRPO	$84.55_{\pm 0.89}$	$\underline{54.11}_{\pm 1.49}$	$\underline{20.57}_{\pm 0.73}$	$\underline{52.27}_{\pm 0.26}$
OPD	$69.25_{\pm 3.10}$	$48.69_{\pm 1.17}$	$18.43_{\pm 1.77}$	$44.93_{\pm 1.36}$
TIP	$53.49_{\pm 0.83}$	$40.86_{\pm 1.30}$	$16.79_{\pm 1.02}$	$36.73_{\pm 0.37}$
RLAD	$\underline{84.70}_{\pm 2.30}$	$53.92_{\pm 0.54}$	$16.57_{\pm 0.70}$	$50.92_{\pm 0.91}$
RAD (ours)	$\mathbf{89.67}_{\pm \mathbf{1.86}}$	$\mathbf{63.51}_{\pm \mathbf{0.34}}$	$\mathbf{24.86}_{\pm \mathbf{1.50}}$	$\mathbf{58.67}_{\pm \mathbf{0.52}}$
Δ vs. best baseline	(+4.97)	(+9.40)	(+4.29)	(+6.40)

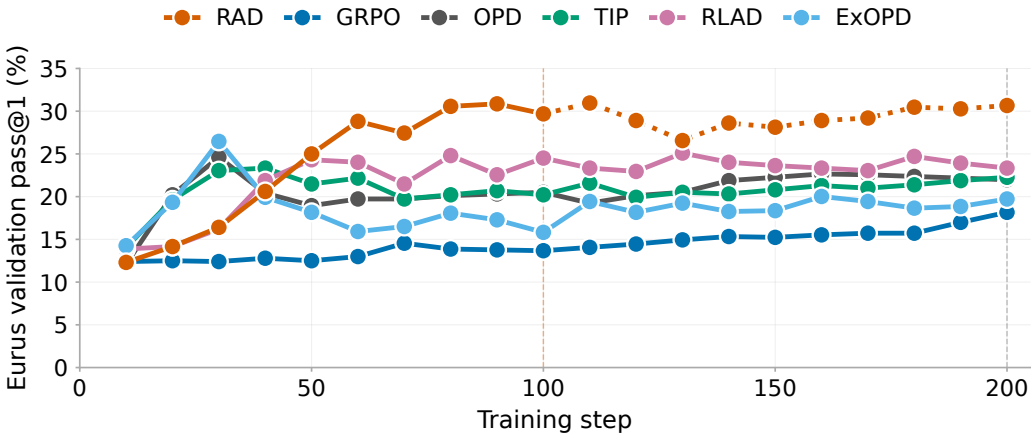

Figure 7: Coding-validation dynamics for Qwen3-1.7B (non-thinking). Each point reports pass@1 over all 1,024 validation problems for one training run per method. Curves stop at the last available evaluation. We compare RAD at step 100 with baselines at step 200; the RAD curve is dotted beyond step 100 and extends to step 200.

Table 9: Qwen3-1.7B (non-thinking) coding-validation pass@1 (%). Step-100 scores are shown by dataset; available step-200 scores are shown overall. Overall weights all 1,024 problems equally. Bold/underline: best/second-best reported value per column, with ties sharing a rank; dashes: unavailable. Parentheses give RAD's gain over the best baseline at the same step (pp, before rounding). RAD at step 100 matches baselines at step 200 in training responses.

	Step 100					Step 200
Method	APPS	CodeContests	Codeforces	TACO	Overall	Overall
GRPO	17.61	17.51	15.45	7.85	13.67	18.16
OPD	28.87	25.99	21.95	11.52	20.51	21.97
TIP	27.46	23.08	<u>26.83</u>	12.57	20.21	22.27
RLAD	<u>37.32</u>	<u>26.79</u>	**31.71**	<u>15.18</u>	<u>24.51</u>	<u>23.34</u>
ExOPD	19.01	20.16	16.26	10.21	15.82	19.73
RAD (ours)	**45.07**	**32.89**	**31.71**	**20.16**	**29.69**	**30.66**
Δ vs. best baseline	$(+7.75)$	$(+6.10)$	$(+0.00)$	$(+4.97)$	$(+5.18)$	$(+7.32)$

C.3 OLMo3-7B Settings and Validation

We train OLMo3-7B-Instruct with OLMo3.1-32B-Instruct as the teacher[2] on the Eurus coding split and evaluate on the same 1,024-problem validation split described in Appendix C.2. We sample one response per problem with temperature 1, top-p 1, and a 32,768-token limit, and report problem-weighted pass@1. Each method uses one training seed, 64 actor prompts with eight responses each, and AdamW with learning rate 10^{-6}. RAD additionally uses 64 meta prompts with eight responses each. We allocate six GPUs to student training and, for distillation methods, two additional GPUs to teacher inference.

RAD uses the same weight settings as the Qwen coding experiment: fixed initial token weight $c = 0$, upper bound 1, a cosine lower-bound schedule from 0 to -1 through step 50, and weight step size $10^5 N_1$ with inverse length scaling and no further normalization. OPD, TIP, and RAD use distillation coefficient $\lambda = 0.1$; TIP uses full-vocabulary reverse KL, and RLAD uses $\alpha = 0.5$.

Table 10: OLMo3-7B-Instruct coding-validation pass@1 (%) over 1,024 Eurus problems, with one response per problem and one training run per method. Bold/underline: best/second-best reported value per column, with ties sharing a rank; dashes: unavailable. Parentheses give RAD's step-100 gain over the best available baseline at the indicated step (percentage points, before rounding).

Method	Step 100	Step 200
GRPO	<u>34.28</u>	**35.84**
OPD	33.20	33.01
TIP	27.44	25.68
RLAD	<u>34.28</u>	<u>33.98</u>
RAD (ours)	**40.23**	—
Δ vs. best baseline	$(+5.96)$	$(+4.39)$

RAD solves 412 of 1,024 validation problems at step 100, compared with 351 for both GRPO and RLAD at step 100 and 367 for GRPO at step 200 (Table 10 and Figure 8). At step 200, OPD, TIP, and RLAD reach 33.01%, 25.68%, and 33.98%, respectively. These single-run results support applying RAD beyond the Qwen student–teacher pairs; they do not isolate the effects of model family and student size.

We also evaluate these checkpoints on all 1,055 LiveCodeBench-v6 tasks using the sampling and native grading protocol of the main coding experiments, with each OLMo checkpoint's native

[2]Model checkpoints: `https://huggingface.co/allenai/Olmo-3-7B-Instruct` and `https://huggingface.co/allenai/Olmo-3.1-32B-Instruct`.

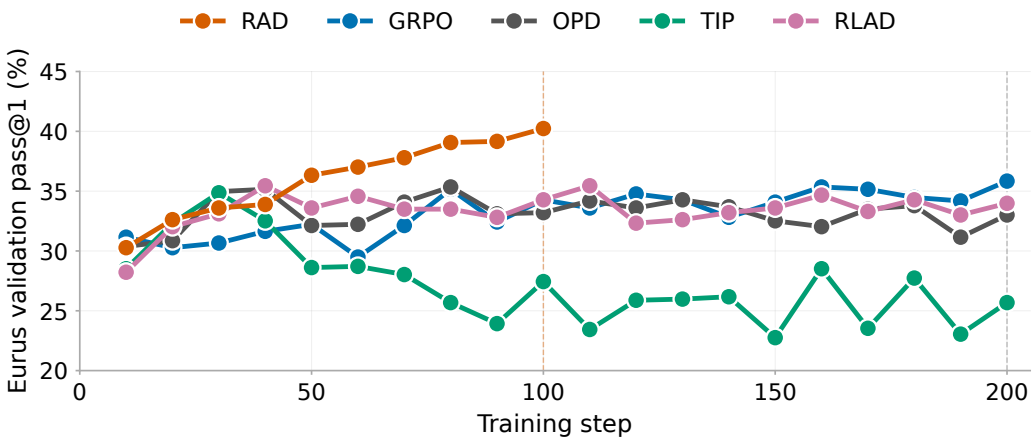

Figure 8: Coding-validation dynamics for OLMo3-7B-Instruct. Each point reports pass@1 over all 1,024 Eurus validation problems. Curves stop at the last available evaluation. We compare RAD at step 100 with all four baselines at step 200.

tokenizer and chat template. Table 8 reports the full results, including the base student and teacher references.

D Preliminary Experiments on Agentic RL Settings

We apply RAD to an agentic software-engineering task. A Qwen3.5-4B [3] policy is trained on R2E-Gym (Jain et al., 2025b) with the OpenHands agent harness provided by R2E-Gym. Each episode presents a repository at a failing commit and lets the policy read, edit, and run code for up to 75 tool turns within a 65,536-token context, generating at most 8,192 tokens per turn with temperature 1, top-p 1, and no top-k truncation; the reward is the hidden test suite's verdict on the final patch. Each update samples 32 tasks with eight attempts each and uses a constant learning rate of 10^{-6}. The teacher is Qwen3.6-27B.[4] The baselines train for one epoch on R2E-Gym, and RAD for half an epoch.

We evaluate on SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024) with three rollouts for each of its 500 tasks under the training scaffold, sampling parameters, and context budget. Its repositories are disjoint from the training corpus. pass@1 is the mean resolved rate over all 1,500 attempts, and pass@3 is the fraction of tasks resolved by at least one of the three attempts.

RAD with negative-weight warmup reaches 50.5% pass@1 and 63.6% pass@3, the best of the five methods (Figure 9). It improves on GRPO by 5.9 points and on the strongest non-RAD baseline, RLAD, by 3.7 points, while training for half as many epochs.

E Computational Cost and Efficiency

E.1 Comparison with Baselines

Table 11 compares total training time for Qwen3-1.7B (non-thinking) on math and OLMo3-7B-Instruct on coding. RAD uses 100 updates and the baselines use 200, matching 102,400 generated training responses. We include checkpoint saves and exclude periodic validation, process startup, and time between training attempts. Each retained update is counted once. The main runs use six H100 GPUs for the student, whereas OPD[2] uses four student GPUs alongside its two-teacher setup. These are measured costs of the experimental configurations: generated response lengths, teacher execution, and GPU allocations also affect step time.

[3] https://huggingface.co/Qwen/Qwen3.5-4B
[4] https://huggingface.co/Qwen/Qwen3.6-27B.

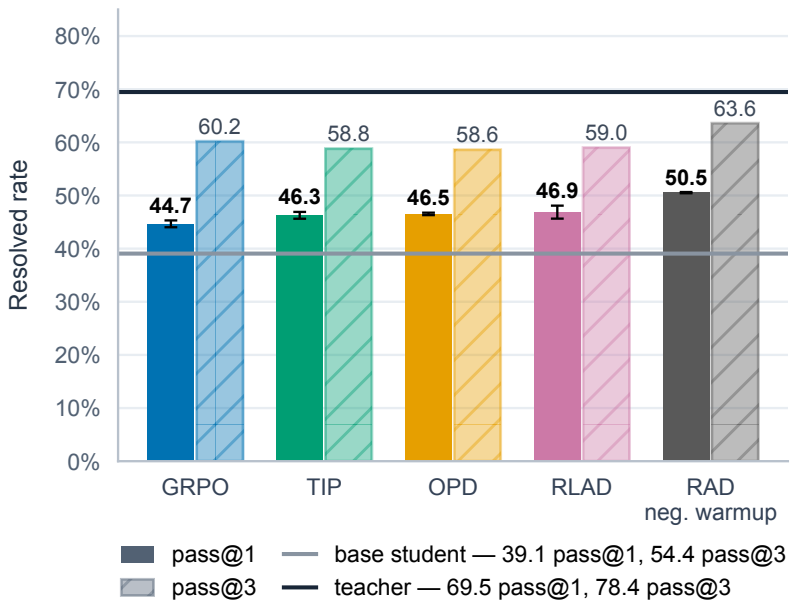

Figure 9: SWE-bench Verified at a 65,536-token context and 75 tool turns. Solid bars show pass@1 with ± 1 standard deviation across the three rollouts; hatched bars show pass@3. Horizontal lines mark the pass@1 of the base student and the teacher.

Table 11: Total training time (hours) for RAD at step 100 and baselines at step 200, matching generated training responses. Checkpoint saves are included; periodic validation, startup, and restart downtime are excluded.

Method	Steps	Student GPUs	Qwen3-1.7B (non-thinking), math	OLMo3-7B-Instruct Coding
OPD	200	6	14.28	136.51
TIP	200	6	34.09	166.79
RLAD	200	6	14.11	123.81
ExOPD	200	6	51.02	—
OPD^2	200	4	33.99	—
RAD	100	6	16.66	101.61

E.2 RAD Stage Breakdown

We profile the first 100 RAD updates for Qwen3-1.7B (non-thinking) on math, which uses eight NVIDIA H100 80GB GPUs: six for the student and its rollouts, and two for teacher inference. Each step generates 512 actor responses and 512 meta responses, with a maximum generation length of 32,768 tokens. Table 12 averages the recorded trainer timers over all 100 steps, including checkpoint saves but excluding periodic validation, job startup, and I/O outside the training-step timer.

Table 12: RAD math training-time breakdown for Qwen3-1.7B (non-thinking), averaged over the same 100 updates as Table 11. Entries give seconds per training step and each stage's share of recorded training-step time. Checkpoint saves are amortized over all steps; validation is excluded and reported separately in the text.

Stage	Seconds	Share
Distillation rollout pipeline	211.1	35.2%
Student log-probabilities and advantages	11.9	2.0%
Lookahead stage	283.2	47.2%
Weight computation and student update	87.5	14.6%
Rollout weight synchronization	5.3	0.9%
Checkpoint saves	0.6	0.1%
Other in-step overhead	0.3	0.1%
Total training step	**599.9**	100%

The distillation rollout pipeline includes generation and overlapping teacher scoring and reward evaluation. The lookahead stage includes the temporary optimizer update, transfer of temporary weights, meta rollouts, reward-gradient computation, and restoration of the student. Weight computation and the committed student update include the exact JVP and projected weight update. These are sequential trainer stages; nested teacher-request and JVP timers are not added again. The recorded timers do not separately resolve the components within the lookahead stage.

Mean training-step time is 10.0 minutes for Qwen3-1.7B (non-thinking) and 16.1 minutes for Qwen3-4B-Instruct-2507, corresponding to 16.66 and 26.79 recorded training hours over 100 steps. Periodic validation adds 5.78 and 9.74 hours, respectively, outside those totals. The lookahead stage accounts for 47.2% and 47.1% of training-step time; weight computation and the student update account for 14.6% and 17.2%. These averages include changing response lengths. The two settings illustrate RAD's executed cost components; they do not isolate model-size effects.

E.3 Math Comparison at Comparable Training Times

We use the cumulative recorded training time of RAD at step 90 for Qwen3-1.7B (non-thinking) as a 12.09-hour reference budget. We use TIP at step 70 and ExOPD at step 50, the first evaluated checkpoints above this budget, for a closer time match. They use 100.9% and 102.6% of the reference budget, respectively. The other baselines use their latest available evaluated checkpoint within the budget. Selection uses time rather than accuracy; we do not interpolate accuracy between checkpoints. Evaluations occur every ten steps and cover the original four benchmarks, so AIME26 and OlympiadBench are excluded from this comparison.

Training time sums the logged durations of completed updates along the retained training trajectory, including checkpoint saves. We count each retained update once and exclude periodic validation, process startup, restart downtime, and discarded attempts. ExOPD logs checkpoint saves outside its step timer, so their durations are added separately. All runs use six H100 GPUs for the student; teacher execution follows each method's implementation. This compares realized training times, which also depend on generated response lengths and implementation details.

Table 13: Math accuracy for Qwen3-1.7B (non-thinking) at comparable training times, relative to RAD at step 90. Accuracy is the mean over AIME24/25 and HMMT2025-Feb/Nov, with standard deviation across 16 sample-position accuracies. Budget used is the fraction of RAD's cumulative training time. Bold/underline mark the best/second-best means; parentheses give RAD's gain over the strongest displayed baseline (pp).

Method	Step	Training (h)	Budget used	Accuracy (%)
***Student: Qwen3-1.7B (non-thinking)**; budget 12.09 h*				
GRPO	290	11.98	99.1%	$26.98_{\pm 2.22}$
OPD	160	11.72	96.9%	$23.23_{\pm 2.20}$
TIP	70	12.19	100.9%	$24.27_{\pm 2.62}$
RLAD	170	12.08	99.9%	$25.10_{\pm 2.67}$
ExOPD	50	12.40	102.6%	$\underline{27.34}_{\pm 2.70}$
RAD (ours)	90	12.09	100.0%	$\mathbf{32.81}_{\pm \mathbf{2.84}}$
Δ vs. best	—	—	—	$(+5.47)$

For Qwen3-1.7B (non-thinking), RAD exceeds the strongest selected baseline, ExOPD, by 5.47 percentage points, despite ExOPD using 2.6% more training time. GRPO's step 290 uses 99.1% of the reference budget.

E.4 Extended GRPO Training

Figure 10 compares Qwen3-1.7B (non-thinking) RAD runs through step 90 with GRPO over a 450-step window. RAD reaches 32.81% on the four math benchmarks and 30.86% on Eurus coding validation at step 90. At step 450, GRPO reaches 33.28% on math and 31.54% on coding. GRPO generates 512 training responses per step, while RAD generates 512 actor and 512 meta responses. Both GRPO endpoints use 2.50 times as many training responses as RAD at step 90.

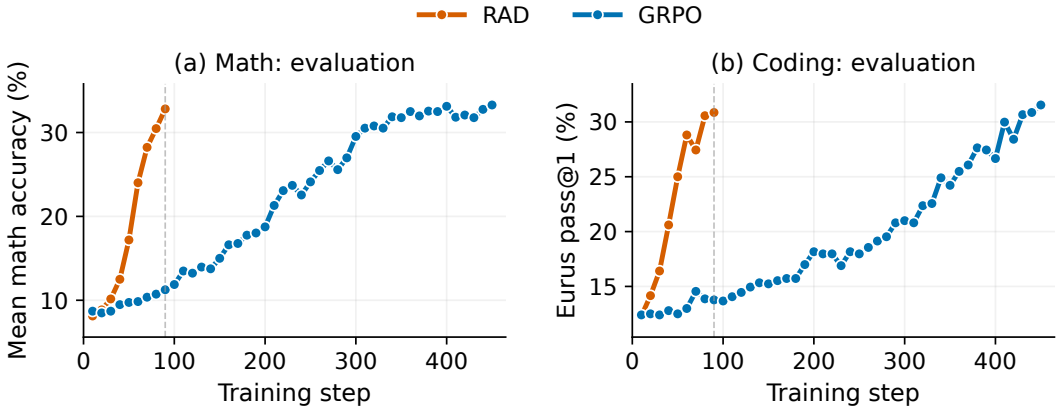

Figure 10: Evaluation during extended training for Qwen3-1.7B (non-thinking). Left: math; right: coding. Math averages AIME24/25 and HMMT2025-Feb/Nov with 16 responses per problem. Coding reports pass@1 on all 1,024 Eurus validation problems. RAD is shown through step 90; GRPO is shown through step 450 in both math and coding. The vertical line marks step 90.

F Gradient of Weighted On-Policy Distillation

Per-token surrogate. For a response sampled from the frozen policy π_{old}, the surrogate used in Equations (1) and (8) is

$$\ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) = \rho_t(\boldsymbol{\theta}) \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\text{T}}(y_t \mid s_t)}, \qquad \rho_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\text{old}}(y_t \mid s_t)}. \tag{9}$$

The sampling policy and prefixes are held fixed. For batch notation, write $\ell_{D,i,t}(\boldsymbol{\theta}; \boldsymbol{y}_i) = \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}_i)$.

Derivation of the per-token gradient. At a fixed prefix s_t, consider the expected weighted reverse KL

$$L_{D,t}(\boldsymbol{\theta}, w_t \mid s_t) = \mathbb{E}_{y_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)} \left[w_t \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \right].$$

Holding w_t fixed when differentiating with respect to $\boldsymbol{\theta}$ gives

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} L_{D,t}(\boldsymbol{\theta}, w_t \mid s_t) &= \nabla_{\boldsymbol{\theta}} \sum_{y_t} \pi_{\boldsymbol{\theta}}(y_t \mid s_t) w_t \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \\
&= \mathbb{E}_{y_t} \left[w_t \left(\log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} + 1 \right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_t \mid s_t) \right] \\
&= \mathbb{E}_{y_t} \left[w_t \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_t \mid s_t) \right] \\
&\quad + \mathbb{E}_{y_t} \left[w_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_t \mid s_t) \right]. \qquad (10)
\end{aligned}$$

This is the per-token, frozen-prefix gradient used in Section 4. The surrogate in Equation (9) realizes the same gradient. At the update point $\pi_{\boldsymbol{\theta}} = \pi_{\mathrm{old}}$, the ratio $\rho_t = 1$ in value but remains differentiable, giving

$$\nabla_{\boldsymbol{\theta}} \ell_{D,i,t}(\boldsymbol{\theta}; \boldsymbol{y}_i) = \left(\log \frac{\pi_{\boldsymbol{\theta}}(y_{i,t} \mid s_{i,t})}{\pi_{\mathrm{T}}(y_{i,t} \mid s_{i,t})} + 1 \right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_{i,t} \mid s_{i,t}).$$

Since the sampling distribution in Equation (1) is fixed, its gradient is simply

$$\nabla_{\boldsymbol{\theta}} L_D(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\mathrm{old}}(\cdot|\boldsymbol{x})} \left[\sum_t w_t(\boldsymbol{y}) \nabla_{\boldsymbol{\theta}} \ell_{D,t}(\boldsymbol{\theta}; \boldsymbol{y}) \right].$$

Student rollouts provide a sampled estimate of this expected gradient.

Action-independent weights and OPD. If w_t does not depend on y_t, the second expectation in Equation (10) vanishes by the zero-mean score identity. The result is w_t times the OPD gradient, with OPD recovered at $w_t = 1$. For an action- or response-dependent weight, this term generally does not vanish. If the weight had explicit differentiable dependence on $\boldsymbol{\theta}$, an additional term

$$\mathbb{E}_{y_t} \left[\log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \nabla_{\boldsymbol{\theta}} w_t \right]$$

would appear; the updated weights in Equation (8) are instead detached during student training, and its normalization is fixed.

Full sequence-level gradient. Equation (10) conditions on a fixed prefix and is therefore a per-token gradient. For comparison, define the full sequence-level objective

$$L_D^{\mathrm{seq}}(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \left[\sum_t w_t(\boldsymbol{y}) \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \right].$$

Unlike Equation (1), its sampling distribution depends on $\boldsymbol{\theta}$. Differentiating the complete trajectory expectation gives

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} L_D^{\mathrm{seq}}(\boldsymbol{\theta}, \boldsymbol{w}) &= \mathbb{E}_{\boldsymbol{y}} \left[\sum_t w_t(\boldsymbol{y}) \nabla_{\boldsymbol{\theta}} \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \right] \\
&\quad + \mathbb{E}_{\boldsymbol{y}} \left[\left(\sum_t w_t(\boldsymbol{y}) \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} \right) \sum_k \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_k \mid s_k) \right].
\end{aligned}$$

The first term supplies the $+1$ contribution because the teacher is fixed and

$$\nabla_{\boldsymbol{\theta}} \log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_t \mid s_t).$$

Combining this with the diagonal $k = t$ contribution after expanding the second term gives

$$w_t(\boldsymbol{y}) \left(\log \frac{\pi_{\boldsymbol{\theta}}(y_t \mid s_t)}{\pi_{\mathrm{T}}(y_t \mid s_t)} + 1 \right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(y_t \mid s_t),$$

which is the frozen-prefix contribution used by the main derivation. The remaining off-diagonal $k \neq t$ terms are the cross-token trajectory-distribution gradients omitted by the fixed-prefix estimator. Our implementation uses this fixed-prefix estimator and refreshes the state distribution with new on-policy rollouts at every iteration.

G Hypergradient through One Optimizer Step

For the fixed batch in Equation (8), let $\boldsymbol{g}(\boldsymbol{w}) = \nabla_{\boldsymbol{\theta}} \widehat{L}_D(\boldsymbol{\theta}, \boldsymbol{w}; \mathcal{B})$. One optimizer step and its evaluation at initial token weight c are

$$\boldsymbol{\theta}^+(\boldsymbol{w}) = \mathcal{A}(\boldsymbol{\theta}, \boldsymbol{g}(\boldsymbol{w})), \qquad \boldsymbol{\theta}^+ = \boldsymbol{\theta}^+(c\mathbf{1}).$$

The batch, original parameters, and current optimizer state are fixed. Using the reward direction $\boldsymbol{p}$ from Equation (6), the weight derivative is

$$\left. \frac{\partial J_R(\boldsymbol{\theta}^+(\boldsymbol{w}))}{\partial w_{i,t}} \right|_{\boldsymbol{w}=c\mathbf{1}} = -\frac{\lambda}{N} \langle \boldsymbol{p}, \nabla_{\boldsymbol{\theta}} \ell_{D,i,t}(\boldsymbol{\theta}; \boldsymbol{y}_i) \rangle = \frac{\lambda}{N} I_{i,t}. \tag{11}$$

The JVP computes these scores together. Their fixed common scaling is absorbed into γ, and we update the weights using Equation (5).

SGD.

For an unclipped SGD step without momentum,

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta} - \eta \boldsymbol{g}, \qquad \boldsymbol{p} = \eta \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+).$$

This gives the SGD score in Equation (4), with the learning rate already included. Momentum or gradient clipping is handled by including that operation in $\mathcal{A}$.

AdamW.

Our experiments use AdamW with global gradient clipping. Let $\widetilde{\boldsymbol{g}}(\boldsymbol{w}) = \mathcal{C}(\boldsymbol{g}(\boldsymbol{w}))$, where $\mathcal{C}$ is the clipping operator; we suppress the dependence on $\boldsymbol{w}$ below. Given the current moments $(\boldsymbol{u}, \boldsymbol{v})$ and update index k, AdamW computes

$$\begin{aligned}
\boldsymbol{u}^+ &= \beta_1 \boldsymbol{u} + (1-\beta_1)\widetilde{\boldsymbol{g}}, & \boldsymbol{v}^+ &= \beta_2 \boldsymbol{v} + (1-\beta_2)\widetilde{\boldsymbol{g}} \odot \widetilde{\boldsymbol{g}}, \\
\widehat{\boldsymbol{u}}^+ &= \frac{\boldsymbol{u}^+}{1-\beta_1^k}, & \widehat{\boldsymbol{v}}^+ &= \frac{\boldsymbol{v}^+}{1-\beta_2^k}, \\
\boldsymbol{\theta}^+(\boldsymbol{w}) &= (1-\eta d)\boldsymbol{\theta} - \eta\, \widehat{\boldsymbol{u}}^+ \oslash \left(\sqrt{\widehat{\boldsymbol{v}}^+} + \epsilon_{\mathrm{A}} \right).
\end{aligned}$$

Here η is the learning rate, d is weight decay, β_1, β_2 are the moment coefficients, and ϵ_{A} is the Adam numerical-stability constant. All vector operations are elementwise; $\odot$ and $\oslash$ denote multiplication and division.

Let D be the Jacobian of $\widehat{\boldsymbol{u}}^+ / (\sqrt{\widehat{\boldsymbol{v}}^+} + \epsilon_{\mathrm{A}})$ with respect to $\widetilde{\boldsymbol{g}}$. Specializing the optimizer-aware reward direction to AdamW gives

$$\boldsymbol{p} = \eta \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}} \right)^{\top} D^{\top} \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+). \tag{12}$$

Substituting Equation (12) gives the AdamW-specialized JVP used in training:

$$\mathbf{I}(\boldsymbol{y}) = -\operatorname{JVP}_{\boldsymbol{\theta}} \left[\boldsymbol{\ell}_D(\boldsymbol{\theta}; \boldsymbol{y}) \right] \left[\eta \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}} \right)^{\top} D^{\top} \nabla_{\boldsymbol{\theta}^+} J_R(\boldsymbol{\theta}^+) \right]. \tag{13}$$

H CUSTOM TRITON KERNEL FOR ATTENTION JVPS

In the `flash_triton` attention implementation, FlashAttention computes the attention output and row log-sum-exp during the model JVP in Equation (7), and a custom Triton rule propagates their directional derivatives. We describe scaled dot-product self-attention with zero dropout and no score softcapping. For one sequence and head,

$$S = QK^\top/\sqrt{d}, \qquad P = \mathrm{softmax}(S), \qquad O = PV, \qquad L_i = \log \sum_j \exp(S_{ij}),$$

where Q, K, V are the queries, keys, and values, and d is the head dimension. Softmax and log-sum-exp include only allowed keys in each row; masked probabilities are zero. Structural masks are fixed during differentiation. Dots denote directional derivatives induced by the parameter direction $\boldsymbol{p}$, with $\dot{Q}, \dot{K}, \dot{V}$ supplied by forward-mode differentiation through the preceding operations. The attention JVP is

$$\begin{aligned} \dot{S} &= (\dot{Q}K^\top + Q\dot{K}^\top)/\sqrt{d}, & a_i &= \sum_j P_{ij}\dot{S}_{ij}, \\ \dot{O} &= P\dot{V} + (P \odot \dot{S})V - a \odot O, & \dot{L} &= a. \end{aligned} \tag{14}$$

Here $\odot$ is elementwise multiplication, with a broadcast across each output row. The term $-a \odot O$ accounts for differentiating the softmax normalizer. Reusing the saved O, L, the kernel reconstructs probabilities one tile at a time and accumulates Equation (14).

Algorithm 2 Tiled attention JVP using the FlashAttention output and log-sum-exp

Input: Q, K, V, tangents $\dot{Q}, \dot{K}, \dot{V}$, saved O, L, sequence boundaries, and attention masks.
Output: Attention-output tangent $\dot{O}$ and log-sum-exp tangent $\dot{L}$.

Initialize $\dot{O}, \dot{L}$ to zero, including padded positions.
for each sequence, query head, and valid query block B in parallel **do**
  Select the corresponding key/value head for grouped-query attention.
  Load $Q_B, \dot{Q}_B, O_B, L_B$; initialize FP32 accumulators $U_B \leftarrow 0$, $a_B \leftarrow 0$.
  for each key block C in the same sequence allowed by the block's masks **do**
    Load $K_C, V_C, \dot{K}_C, \dot{V}_C$.
    $S_{BC} \leftarrow Q_B K_C^\top/\sqrt{d}$.
    $\dot{S}_{BC} \leftarrow (\dot{Q}_B K_C^\top + Q_B \dot{K}_C^\top)/\sqrt{d}$.
    $P_{BC} \leftarrow \exp(S_{BC} - L_B)$, broadcasting L_B over keys.
    Set masked entries of P_{BC} to zero.
    $a_B \leftarrow a_B + \mathrm{rowsum}(P_{BC} \odot \dot{S}_{BC})$.
    $U_B \leftarrow U_B + P_{BC}\dot{V}_C + (P_{BC} \odot \dot{S}_{BC})V_C$.
  end for
  Store $\dot{O}_B \leftarrow U_B - a_B \odot O_B$ and $\dot{L}_B \leftarrow a_B$ for valid queries.
end for

Query blocks stay within sequence boundaries, so packed sequences cannot attend to one another. Causal, padding, and sliding-window masks use the same valid query–key pairs as the primal; grouped-query heads share their key/value tensors without explicitly repeating them. Our training launchers use 16-query blocks, with Triton autotuning key-block sizes of 16 or 32. Probability factors and tangent accumulators remain in FP32; the products with V and $\dot{V}$ use `tf32x3` tensor-core precision. The kernel stores $\dot{O}$ in the primal output's dtype and $\dot{L}$ in FP32. It avoids storing full attention or attention-tangent matrices, while full-attention arithmetic remains quadratic in sequence length. The rule analytically differentiates the attention equations up to floating-point error.

H.1 LAYER-STREAMED JVP EXECUTION

The `olmo3_layer_streamed` backend evaluates the embedding, decoder layers, final normalization, and output projection in sequence. For each parameter group, the FSDP executor gathers

its parameters and the corresponding components of $\boldsymbol{p}$, propagates both activations and activation tangents, and releases the gathered tensors after use. It can prefetch the next group, so gathered parameter and direction storage is bounded by the current and prefetched groups. Activation states are retained for a bounded set of microbatches at a time, with microbatch grouping synchronized across ranks to preserve the order of collective operations.

Within each OLMo decoder layer, the backend uses the same FlashAttention primal and Triton attention tangent as `flash_triton` (Algorithm 2), including the masks for full and sliding-window attention. Forward-mode differentiation propagates the chain-rule JVP through successive layers, reducing the memory needed for gathered parameters and their direction while retaining the token-score computation in Equation (7).