OPTIMIZE MORE, SAMPLE LESS: EFFICIENT ON-POLICY DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy distillation (OPD) improves a student model by repeatedly sampling from its current policy and distilling feedback from a stronger teacher. However, standard OPD tightly couples sampling and optimization: after each round of rollouts, the student is updated only briefly before new trajectories are collected. This can be inefficient when on-policy sampling and teacher inference are expensive, while optimization on already collected trajectories is comparatively cheap. We revisit this design choice and ask a simple question: should we optimize more before sampling again? We introduce a simple modification to OPD that performs multiple optimization steps on each batch of on-policy trajectories before refreshing the rollout distribution. By increasing the number of optimization steps per rollout batch, the student moves closer to the teacher within each rollout round and extracts more learning signal from every sampled trajectory. On DeepSeek-R1-Distill-Qwen-1.5B across a suite of mathematical reasoning benchmarks, additional optimization reaches a given performance level with substantially fewer rollouts and teacher generations. Compared with vanilla OPD, taking 128 optimization steps per rollout batch achieves a $2.07\times$ wall-clock speedup and uses 64.9% fewer FLOPs, while maintaining or improving final performance. Surprisingly, aggressive reuse of on-policy data remains effective well beyond the conventional one-update-per-rollout regime. Our results suggest that frequent resampling is not necessary for effective on-policy distillation: when sampling is expensive, a simple principle works remarkably well—optimize more, sample less.

1 INTRODUCTION

On-policy distillation (OPD) offers an effective and economical way to transfer the capabilities of a strong teacher to a smaller language model (Agarwal et al., 2024; Lu & Thinking Machines Lab, 2025). The student generates its own responses and learns from the teacher's next-token predictions along those responses, receiving dense supervision without requiring ground-truth answers or a task-specific reward model. This approach can deliver strong reasoning performance at a low post-training cost: Qwen3 reports that OPD outperforms direct reinforcement learning from the same initialization using roughly one-tenth of the GPU hours (Qwen Team, 2025).

Despite these successes, OPD remains costly for long reasoning sequences. The student must generate long trajectories, and the teacher must process these long sequences to provide next-token supervision (Zhang et al., 2026a; Wu et al., 2026a). Both costs recur throughout training as new batches are collected. Nor does this computation always translate into substantial improvements in the student. Further improving an already capable thinking model can be difficult: standard OPD has shown only modest gains even with a stronger teacher (He et al., 2026).

To resolve this issue, we revisit how OPD uses each batch of trajectories and teacher feedback. Practical OPD applies a stop-gradient to the sampling policy, treating the rollout distribution as fixed during differentiation. We therefore formulate its update through a decoupled objective $f(\varphi, \theta)$: the sampler φ determines which states are visited, while the actor θ learns to match the teacher on those states. Vanilla OPD first generates a rollout batch with the sampler, then trains the actor on that batch for one optimizer step. It synchronizes the sampler with the updated actor before generating the next batch. The underlying distillation task, however, is to align the student and teacher distributions on sampled states. This suggests that strict on-policy sampling may not be necessary at every update: the

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

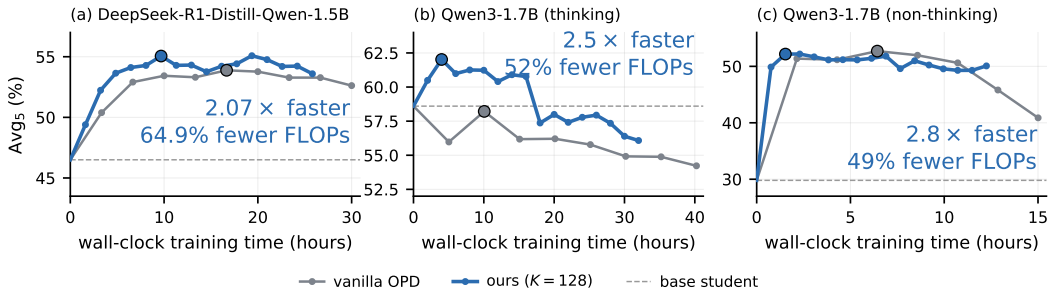

Figure 1: **Accuracy against wall-clock training time, vanilla OPD and ours (K=128), on three student–teacher pairs.** Traj replay achieves comparable or better best-checkpoint accuracy in less training time. Annotations report wall-clock speedups and FLOPs reductions over 256 optimizer steps, with FLOPs computed under our parameter-count model. Time scales optimizer steps by each run's per-step time. Large markers are the best checkpoints; the dashed line is the base student.

actor can continue learning to match the teacher on states already collected. We therefore suspect that a single update does not make full use of the teacher's supervision on each batch. To investigate, we track the loss on a fixed batch over 32 epochs in each of our three student settings. The first update reduces disagreement only slightly, whereas subsequent updates continue to lower it substantially (Section 2.3). This observation motivates our hypothesis:

Can OPD reach higher accuracy in less training time by optimizing more before sampling again?

We test this hypothesis with *trajectory replay*: we sample a batch and cache the teacher's top-k support and corresponding probabilities at each visited state, then take K optimizer steps on the same trajectories before syncing the sampler. Overfitting is a natural concern with repeated data reuse, yet we observe no clear signs of it in our trajectory replay experiments, even without an additional policy-KL penalty (Section 4.2). A plausible explanation is that our top-k reverse KL matches the teacher's distribution rather than a single observed continuation: its optimum retains the teacher's uncertainty over candidate tokens instead of concentrating probability on one token (Sections 2.3 and 4.2).

We evaluate this recipe on three student–teacher pairs spanning two student models, three teachers, and two prompt sets, covering both thinking and non-thinking settings. Figure 1 shows that traj replay achieves comparable or better best-checkpoint accuracy at substantially lower training cost. With 128 updates per batch and the same total number of student updates, it uses 128 times fewer generated and teacher-scored sequences and reduces FLOPs by 49%–64.9%. Qwen3-1.7B training is 2.5× faster in thinking mode and 2.8× faster in non-thinking mode. The largest accuracy gain is on the thinking student, where traj replay reaches 62.0% mean accuracy across five reasoning benchmarks, 3.8 points above vanilla OPD and 3.4 points above the base model. It outperforms vanilla OPD at every evaluated checkpoint in this setting, improving a student that vanilla OPD fails to improve over its base (Section 4.1).

Our contributions are twofold.

1. **A coordinate-wise formulation and a simple trajectory replay recipe.** We decouple the OPD objective by separating how training states are generated from how the student learns to match the teacher on those states. This view motivates our hypothesis that further optimization on each rollout batch can improve training efficiency. Thus, we propose to take multiple optimizer steps on each fixed batch of rollouts before collecting a new batch.
2. **Strong empirical performance at lower cost.** Across three student–teacher pairs, traj replay achieves comparable or better best-checkpoint accuracy with 128× fewer generated and teacher-scored sequences and 49%–64.9% fewer FLOPs. It makes Qwen3-1.7B training 2.5–2.8× faster and improves its thinking-mode AVG_5 by 3.8 points over vanilla OPD (Figure 1 and Section 4.1).

2 ON-POLICY DISTILLATION, REVISITED

2.1 FROM THE OPD OBJECTIVE TO THE IMPLEMENTED LOSS

OPD trains a student on teacher feedback along its own rollouts (Agarwal et al., 2024; Lu & Thinking Machines Lab, 2025). For a student π_θ, a frozen teacher μ, and a prompt $x \sim \mathcal{D}$, a rollout y visits states $s_t = (x, y_{<t})$. Define $\ell(\theta; s) = D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \| \mu(\cdot \mid s))$ for the next-token divergence at a fixed state. For reverse-KL OPD, the ideal objective compares the student and teacher distributions over complete responses (Gu et al., 2026). By the autoregressive chain rule,

$$\mathcal{L}^{\text{ideal}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\big[D_{\mathrm{KL}}(\pi_\theta(\cdot \mid x) \| \mu(\cdot \mid x))\big] = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x)}\Big[\sum_{t=1}^{|y|} \ell(\theta; s_t)\Big]. \tag{1}$$

In practice, we replace the full-vocabulary divergence with a loss on the teacher's top-k tokens (Fu et al., 2026). For a batch $B = \{(x_i, y_i)\}_{i=1}^N$ sampled from the current student, the teacher scores each visited prefix $s_{i,t} = (x_i, y_{i,<t})$. At each such prefix s, we retain the teacher's k highest-probability tokens, $S_k(s) = \mathrm{TopK}_k(\mu(\cdot \mid s))$, and restrict both distributions to this shared set, renormalizing each to sum to one:

$$\bar{\pi}_\theta(a \mid s) = \frac{\pi_\theta(a \mid s)}{\sum_{a' \in S_k(s)} \pi_\theta(a' \mid s)}, \quad \bar{\mu}(a \mid s) = \frac{\mu(a \mid s)}{\sum_{a' \in S_k(s)} \mu(a' \mid s)}, \quad a \in S_k(s). \tag{2}$$

The local loss is $\ell_k(\theta; s) = D_{\mathrm{KL}}(\bar{\pi}_\theta(\cdot \mid s) \| \bar{\mu}(\cdot \mid s))$: it matches the relative probabilities of the retained candidates and discards the mass outside this set (Huang & Wei, 2026). Holding the sampled prefixes fixed, we differentiate the resulting batch loss through the student:

$$\widehat{\mathcal{L}}_{B,k}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{|y_i|} \ell_k(\theta; s_{i,t}), \quad \hat{g}_{\text{top-}k}(\theta) = \nabla_\theta \widehat{\mathcal{L}}_{B,k}(\theta). \tag{3}$$

The population surrogate corresponding to this update is:

$$\mathcal{L}_k^{\text{practical}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_{\mathrm{sg}[\theta]}(\cdot|x)}\Big[\sum_{t=1}^{|y|} \ell_k(\theta; s_t)\Big]. \tag{4}$$

Replacing ℓ_k with ℓ gives the full-vocabulary counterpart, denoted by $\mathcal{L}^{\text{practical}}$. Full-vocabulary OPD (Agarwal et al., 2024) and sampled-token OPD (Lu & Thinking Machines Lab, 2025) also hold the sampled prefixes fixed during backpropagation; they differ in how the local loss $\ell(\theta; s)$ and its gradient are computed or estimated.

2.2 DECOUPLING THE SAMPLER AND THE ACTOR IN OPD: A COORDINATE-WISE VIEW

The practical objective in Equation (4) holds the rollout policy fixed while updating the student. To expose the optimization problem within this update, introduce an auxiliary function with separate parameters for the *sampler* φ and the *actor* θ:

$$f(\varphi, \theta) = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\varphi(\cdot|x)}\Big[\sum_{t=1}^{|y|} \ell(\theta; s_t)\Big] = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\varphi(\cdot|x)}\Big[\sum_{t=1}^{|y|} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid s_t) \| \mu(\cdot \mid s_t)\big)\Big]. \tag{5}$$

The ideal objective is $\mathcal{L}^{\text{ideal}}(\theta) = f(\theta, \theta)$. Differentiating both the local losses and the rollout probabilities gives

$$\nabla_\theta \mathcal{L}^{\text{ideal}}(\theta) = \underbrace{\mathbb{E}_{\substack{x\sim\mathcal{D}\\ y\sim\pi_\theta(\cdot|x)}}\Big[\sum_{t=1}^{|y|} \nabla_\theta \ell(\theta; s_t)\Big]}_{\substack{\text{practical update}\\ \partial_2 f(\varphi,\theta)|_{\varphi=\theta} = \nabla_\theta \mathcal{L}^{\text{practical}}(\theta)}} + \underbrace{\mathbb{E}_{\substack{x\sim\mathcal{D}\\ y\sim\pi_\theta(\cdot|x)}}\Big[\nabla_\theta \log \pi_\theta(y \mid x) \sum_{t=1}^{|y|} \ell(\theta; s_t)\Big]}_{\substack{\text{effect of changing the visited states}\\ \partial_1 f(\varphi,\theta)|_{\varphi=\theta}}}. \tag{6}$$

Here ∂_1 and ∂_2 denote gradients with respect to the sampler and actor, respectively; Section A gives the derivation. Practical OPD drops the second term to reduce gradient variance: this policy-gradient

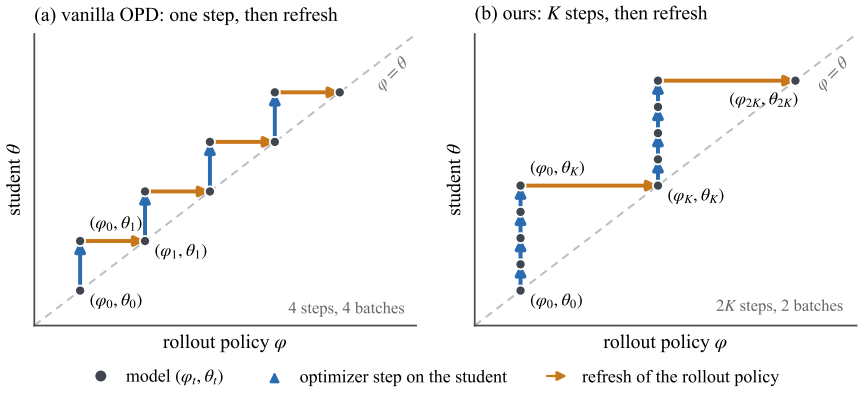

Figure 2: **How the rollout policy φ and the student θ move.** Each vertical arrow updates the actor with the sampler held fixed; each horizontal arrow syncs the sampler with the actor, followed by fresh rollouts and teacher scoring. (a) Vanilla OPD returns to the diagonal after every actor update and pays for a new batch before the next one. (b) Ours takes K optimizer steps on the same batch before returning to the diagonal and collecting new data.

term couples each sampled action to losses at future states, which can make its Monte Carlo estimate noisy on long rollouts (Fu et al., 2026).

Therefore, practical OPD consists of an actor update followed by a sampler sync. The actor update changes θ using teacher feedback on a fixed rollout distribution; the sync changes φ to the updated actor for the next batch. In the decoupled coordinates, the vanilla schedule is

$$(\varphi_t,\theta_t) \xrightarrow[\varphi \text{ fixed}]{\text{actor update}} (\varphi_t,\theta_{t+1}) \xrightarrow[\theta \text{ fixed}]{\text{sampler sync}} (\theta_{t+1},\theta_{t+1}). \tag{7}$$

The first transition optimizes $f(\varphi_t,\theta)$ by updating only the actor; a gradient step with learning rate η gives $\theta_{t+1} = \theta_t - \eta\,\partial_2 f(\varphi_t,\theta_t)$. The second changes the sampling distribution rather than minimizing f over φ. Vanilla OPD performs one actor update before syncing back to the diagonal $\varphi=\theta$ (Figure 2a).

2.3 One update leaves substantial room for learning

Returning to the formulation $f(\varphi,\theta)$, our aim is to align the student with the teacher on states visited by the sampler π_φ. We suspect that a single optimization step is insufficient to fully learn from the teacher's supervision on these states. We therefore propose holding φ fixed and taking multiple actor steps on the same batch before syncing the sampler (Figure 2b), so that the student can fit the teacher more closely on the states already sampled.

To assess how much learning remains after the first update, we track the loss over 32 epochs on a fixed batch for each of the three student settings (Figure 3). Each epoch is one optimizer step over the entire batch. The first epoch reduces the loss only slightly in all three settings, whereas subsequent epochs continue to lower it substantially on the same batch. Vanilla OPD therefore replaces the batch while much of its disagreement with the teacher can still be reduced. This observation motivates the following hypothesis.

Hypothesis. OPD can learn more efficiently by fitting each batch more thoroughly before collecting the next one. We therefore take K optimizer steps on the same batch before refreshing the sampler (Figure 2b), amortizing sampling and teacher scoring over multiple updates.

A natural concern is overfitting: as in SFT and other supervised learning, repeatedly fitting a fixed batch may lower training loss without improving generalization. In our experiments, however, traj replay works well without an additional policy-KL penalty; adding one does not improve accuracy (Figure 7). The supervision provides a plausible explanation. Unlike hard-target SFT, our top-k reverse KL matches the teacher's distribution over candidate tokens rather than a single observed

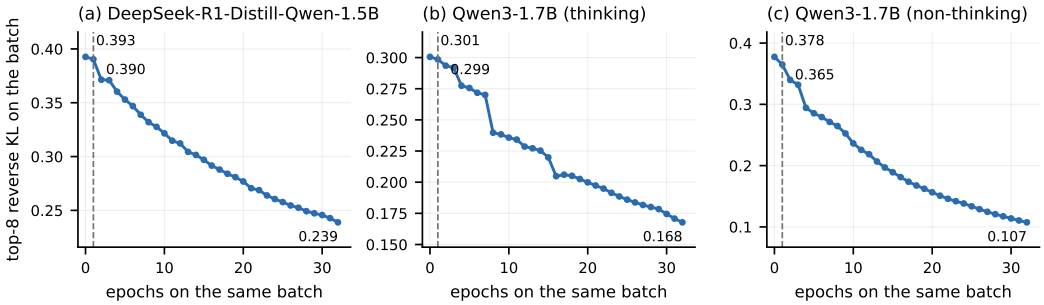

Figure 3: **A single update leaves most of the batch loss unchanged.** Training loss on a fixed batch for (a) DeepSeek-R1-Distill-Qwen-1.5B, (b) Qwen3-1.7B (thinking), and (c) Qwen3-1.7B (non-thinking). Each epoch is one optimizer step over the entire batch; epoch 0 is before training. The dashed line marks the one-update schedule of vanilla OPD.

Algorithm 1 Traj replay: OPD with K optimizer steps per rollout batch; vanilla OPD is $K = 1$

input student π_θ, teacher μ, prompt set $\mathcal{D}$, prompts per batch M, rollouts per prompt G, optimizer steps per rollout batch K, number of rollout steps R
for rollout step $r = 1, \ldots, R$ **do**
  set the rollout policy to the current student: $\varphi \leftarrow \theta$
  sample one batch: draw prompts $x_1, \ldots, x_M \sim \mathcal{D}$ and, for each prompt, G rollouts $y_{i,1}, \ldots, y_{i,G} \sim \pi_\varphi(\cdot \mid x_i)$; $B \leftarrow \{(x_i, y_{i,g})\}$, a batch of $N = MG$ rollouts
  score B once with the teacher: for every state $s_{i,t}$ in B, store $S_k(s_{i,t})$ and $\bar{\mu}(\cdot \mid s_{i,t})$ of Equation (2)
  for $j = 1, \ldots, K$ **do**
    $\theta \leftarrow \mathrm{Opt}\big(\theta, \hat{g}_{\text{top-}k}(\theta)\big)$, one optimizer step on the same batch B (Equation (3))
  end for
end for

continuation (Fu et al., 2026). Its pointwise minimum is attained at $\bar{\pi}_\theta(\cdot \mid s) = \bar{\mu}(\cdot \mid s)$, retaining the teacher's uncertainty within $S_k(s)$ rather than driving probability onto one token.

2.4 Our algorithm

Traj replay is simple: sample one batch, take K actor updates on it with φ fixed, then sync the sampler (Figure 2b). Algorithm 1 gives the procedure, where $\mathrm{Opt}(\theta, g)$ denotes one optimizer step (e.g., Adam). Each update uses the entire batch in our main experiments, so K counts epochs; $K = 1$ recovers vanilla OPD.

For our teacher top-k loss, we cache the support $S_k(s)$ and the teacher's log-probabilities at each prefix when scoring the batch. All K updates reuse these cached values, requiring no further teacher scoring or support selection; only the student's probabilities are recomputed.

3 Related work

Efficient on-policy distillation. Recent work reduces OPD cost by distilling only reasoning prefixes or progressively increasing the rollout horizon (Zhang et al., 2026a;b). Speculative knowledge distillation improves training trajectories by interleaving student proposals with teacher corrections (Xu et al., 2025). Prior work focuses on the distillation loss or the choice of training trajectories. We instead study how many times to replay each batch: we retain the teacher top-k loss of Fu et al. (2026) and take multiple optimizer steps on the same trajectories.

Offline and asynchronous distillation. Other work also relaxes strict on-policy sampling, but targets different sources of cost. Lightning OPD and its cross-teacher extension (Wu et al., 2026a;b) focus on eliminating live teacher serving by collecting rollouts and teacher scores offline. Their

training loop takes one update on a minibatch before moving to another minibatch, rather than repeatedly optimizing the same trajectories for many consecutive steps. AsyncOPD (Kang et al., 2026) focuses on pipeline throughput and gradient estimation under policy lag, consuming queued trajectories in a single training pass. These schedules allow the student to change between trajectory generation and training; this policy mismatch is distinct from traj replay. We explicitly retain each sampled batch for K consecutive optimizer steps, so every trajectory and its cached teacher feedback support multiple updates before we collect new rollouts. Our focus is how much learning can be extracted from each batch, and how this reduces the sampling and teacher-scoring cost per update.

Multi-step optimization and traj replay. Taking several optimization steps before collecting new data is established in RL. PPO optimizes its surrogate over multiple epochs on each rollout batch (Schulman et al., 2017), and Arnal et al. (2026) study how replay ratio and buffer size trade off generation cost, data staleness, and learning performance in LLM training. In supervised learning, data echoing reuses outputs of expensive input-pipeline stages to amortize data loading and preprocessing (Choi et al., 2019). Our schedule applies the same principle to student generation and teacher scoring, while periodically refreshing the data distribution as the student learns. Proximal point methods offer a related inner–outer structure, repeatedly solving proximity-regularized subproblems (Rockafellar, 1976; Parikh & Boyd, 2014). We share the principle of repeated optimization, but study OPD's distribution-matching objective. Unlike PPO or proximal point methods, our updates use neither policy-ratio clipping nor an additional proximity penalty, yet extensive traj replay shows little sign of overfitting in our experiments (Sections 2.3 and 4.2).

4 Experiments

Setup. We train three student–teacher pairs (Table 1): DeepSeek-R1-Distill-Qwen-1.5B (DeepSeek-AI, 2025) from Skywork-OR1-Math-7B (He et al., 2025) on OpenThoughts3-Math prompts (Guha et al., 2025); Qwen3-1.7B (Qwen Team, 2025) from Qwen3-4B-Thinking-2507 on DAPO-Math-17k prompts (Yu et al., 2025); and Qwen3-1.7B with thinking disabled from Qwen3-30B-A3B-Instruct-2507, also on DAPO-Math-17k. The first two students train and are evaluated in thinking mode. Vanilla OPD and ours share every setting except K; the hyperparameters are listed in Section B.

Evaluation. We evaluate on AIME24, AIME25, AIME26, AMC23, and MATH-500, and report the mean accuracy across these benchmarks as AVG_5. Accuracy is averaged over multiple responses per problem, scored using the training framework's boxed-answer grader. We evaluate checkpoints every 16 optimizer steps. Full evaluation details are given in Section B.

4.1 Main result: strong students on three pairs at a fraction of the rollouts

Comparable accuracy at lower cost. Our method achieves comparable or better best-checkpoint accuracy than vanilla OPD across all three student–teacher pairs (Table 1). Figure 4 compares accuracy over training at matched optimizer steps. The late-training accuracy drops are not unique to traj replay: they also occur with vanilla OPD and have been reported in prior OPD studies (Luo et al., 2026; Li et al., 2026). The speedup comes from replaying each rollout batch and its cached teacher feedback for 128 updates. Over 256 optimizer steps, vanilla OPD generates and scores 256 batches, whereas ours needs only two (Figure 5a). Both methods perform the same number of student updates; traj replay removes most of the rollout and teacher-scoring work between them. This reduces FLOPs by 49%–64.9% under our parameter-count accounting (Figure 5b). For Qwen3-1.7B, active training time falls from 80.5 to 32.0 hours in thinking mode (2.5× faster) and from 34.3 to 12.3 hours in non-thinking mode (2.8× faster; Figure 5c).

Substantial gains in thinking mode. Beyond the efficiency gains, traj replay improves Qwen3-1.7B in thinking mode to 62.0 AVG_5, 3.8 points above vanilla OPD and 3.4 points above the base model (Table 1). Our method leads vanilla OPD at every evaluated checkpoint (Figure 4). Improving this model with OPD is nontrivial: vanilla OPD does not improve over the base in our experiments, and He et al. (2026) also report only modest gains from standard OPD with the same student–teacher pair. Their study evaluates several OPD variants, including Cal-OPD, with a best reported AIME24 score of 45.0 among distillation methods, compared with our 51.0. Evaluation protocols differ,

Table 1: **Base student, vanilla OPD and our recipe (K=128 optimizer steps per rollout batch) on three student–teacher pairs, each at its best checkpoint.** Per-benchmark accuracy and AVG$_5$ at the checkpoint with the highest AVG$_5$ on the 16-step grid. The best result in each column within a pair is in bold; shaded rows are our method.

Model	AIME24	AIME25	AIME26	AMC23	MATH500	AVG$_5$
Student: DeepSeek-R1-Distill-Qwen-1.5B Teacher: Skywork-OR1-Math-7B						
Base	30.2	24.0	21.0	73.1	84.0	46.5
Vanilla OPD	40.7	31.1	28.1	**81.4**	88.0	53.9
Ours (K=128)	**43.0**	**31.3**	**32.1**	81.0	**88.1**	**55.1**
Student: Qwen3-1.7B (thinking) Teacher: Qwen3-4B-Thinking-2507						
Base	47.8	35.1	36.2	84.6	89.0	58.6
Vanilla OPD	41.3	36.5	38.1	85.9	89.3	58.2
Ours (K=128)	**51.0**	**39.4**	**42.9**	**86.7**	**90.1**	**62.0**
Student: Qwen3-1.7B (non-thinking) Teacher: Qwen3-30B-A3B-Instruct-2507						
Base	14.2	9.0	9.0	45.3	71.7	29.8
Vanilla OPD	**38.5**	29.0	**30.6**	78.1	87.5	**52.7**
Ours (K=128)	36.3	**30.2**	28.3	**78.3**	**87.9**	52.2

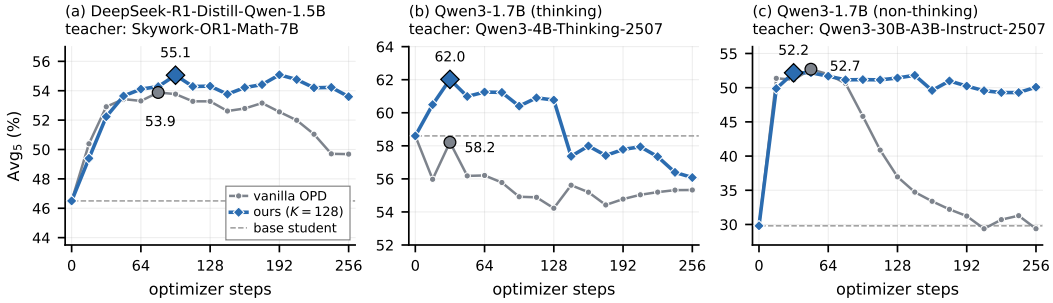

Figure 4: **Accuracy over training for vanilla OPD and ours (K=128) on the three pairs.** Each panel plots AVG$_5$ against optimizer steps; the large markers are the best checkpoints reported in Table 1, and the dashed line is the base student. The same curves broken down by benchmark are in Figure 9.

however, and we have not re-evaluated these methods under our setup; the cross-paper comparison is therefore indicative rather than controlled.

4.2 Ablation Study

In this section we want to understand what makes traj replay work: how long a batch can be replayed before it stops paying, whether the gap to vanilla OPD is just a matter of its learning rate, whether traj replay needs the regularization that RL relies on when it trains on a stale batch, and whether the gain comes from traj replay or prompt replay. Each is settled by one experiment, with DeepSeek-R1-Distill-Qwen-1.5B as the student unless another is named.

Effect of updates per rollout batch. Figure 6a is drawn by fixing everything else (the prompts, the batch of 448 sequences, the learning rate and a budget of 256 optimizer steps) and varying only K, the number of steps taken on each rollout batch, over $\{1, 8, 64, 128\}$. As K increases, rollout and teacher-scoring costs decrease, while accuracy remains comparable or improves slightly.

The gap is not a learning-rate effect. We test whether tuning vanilla OPD's learning rate can close the accuracy gap observed in the main experiments. Figure 6b compares vanilla OPD at learning rates 10^{-6}, 2×10^{-6}, 5×10^{-6} and 10^{-5} with our method at 10^{-5}. Across the tested learning rates, vanilla OPD shows no clear accuracy advantage over our method.

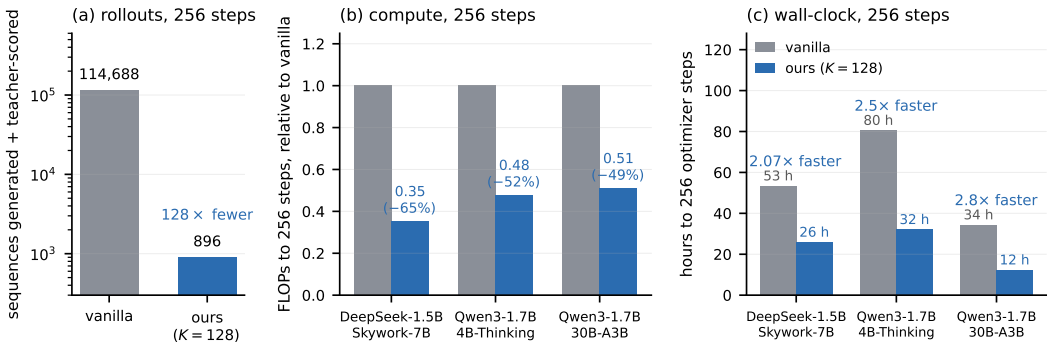

Figure 5: **Cost of training for 256 optimizer steps, vanilla OPD against ours (K=128).** Over those steps vanilla draws a fresh rollout batch and scores it with the teacher at every step; ours does so once per 128 steps. The three panels show what that difference amounts to: (a) the number of sequences generated and teacher-scored; (b) the FLOPs of generation, teacher scoring and training together, relative to vanilla, for each student; (c) the wall-clock time.

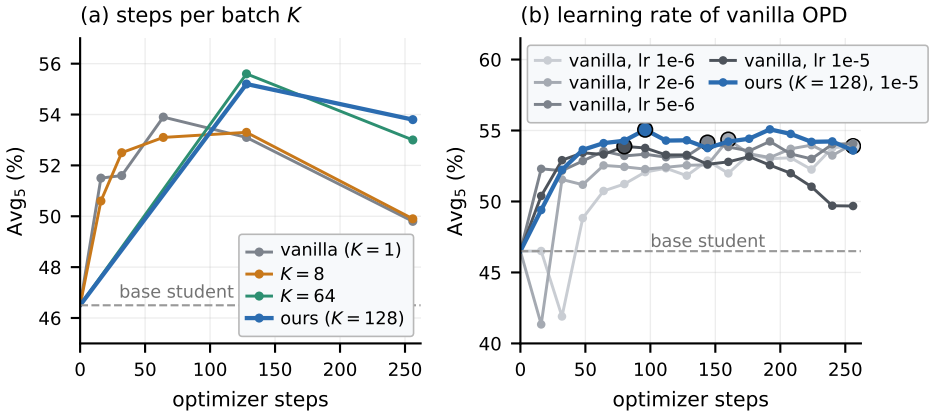

Figure 6: **Ablations with DeepSeek-R1-Distill-Qwen-1.5B.** (a) Accuracy against optimizer steps when only K, the number of optimizer steps taken on each rollout batch, is varied; the dashed line is the base student, and the per-benchmark version is Figure 10. (b) Vanilla OPD at four learning rates against ours; the large markers are the best checkpoints.

Traj replay does not need regularization. In RL, and in proximal policy optimization in particular, taking many steps on one batch is only safe behind regularization, a KL penalty or trust region and importance weights, because the objective would otherwise be overfitted to a stale batch. We added a KL penalty to traj replay and found it not needed. Figure 7 shows the KL penalty on two students: it lowers the drift of the student from the policy that generated its batch (panels b and d) and changes accuracy by nothing (panels a and c). Without the penalty, drift reaches 0.16 and 0.21 nats per token after 128 optimizer steps on one batch. Our inner distillation problem aligns the student with the teacher on a fixed set of sampled states, rather than maintaining strict on-policy sampling at every update. The student can therefore move substantially away from the sampler while still fitting the same teacher targets.

Prompt replay is not enough. We compare traj replay with prompt replay, which fixes the prompts and sampling policy for 128 steps but generates fresh trajectories before each update. Prompt replay still requires rollout generation and teacher scoring at every step. Figure 8 shows that it also underperforms traj replay. For DeepSeek-R1-Distill-Qwen-1.5B, it initially matches vanilla OPD and our method, but accuracy drops sharply after the sampler is updated at step 128 (see Figure 11 for per-benchmark results). For Qwen3-1.7B (thinking), it is below both methods and the base model

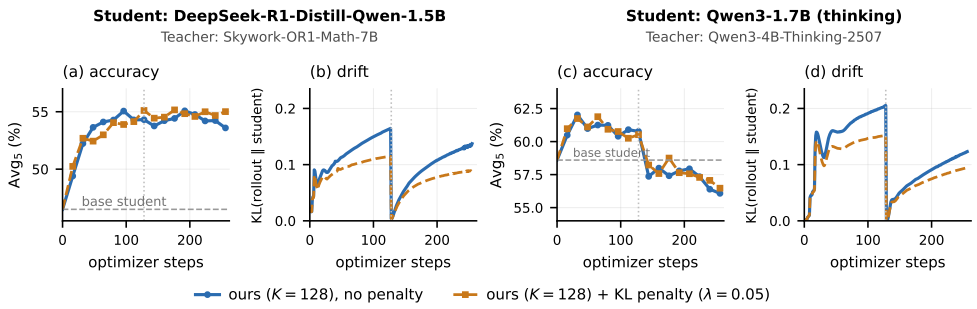

Figure 7: **Traj replay with and without a KL penalty, for two students.** (a, c) Accuracy against optimizer steps with and without the k_3 penalty (λ=0.05) toward the previous pass; the dotted line marks the resample at step 128. (b, d) The same runs' drift from the policy that generated their batch, $D_{\mathrm{KL}}(\pi_{\text{sampling}} \| \pi_\theta)$ measured on the batch's tokens.

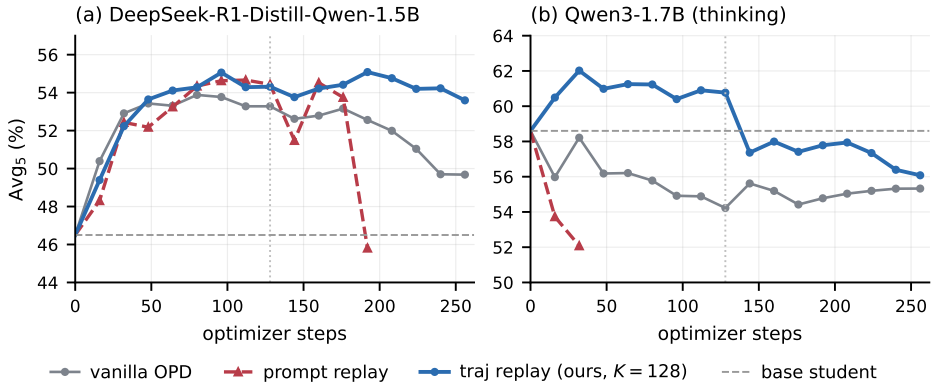

Figure 8: **Prompt replay versus traj replay.** AVG_5 on the 16-step grid for (a) DeepSeek-R1-Distill-Qwen-1.5B and (b) Qwen3-1.7B (thinking). Vanilla OPD samples fresh trajectories from the current student at every step. Prompt replay holds the sampling policy for 128 steps but samples fresh trajectories from it at every step: the base model for the first 128 steps, then the step-128 student (run through step 192 on DeepSeek-R1-Distill-Qwen-1.5B; steps 16 and 32 only on Qwen3-1.7B). Traj replay (ours) keeps the same trajectories for 128 steps. Dotted line: the 128-step boundary; dashed line: the base student.

from the first checkpoint. These results show that keeping the sampler fixed is not enough: the gains depend on repeatedly optimizing on the same sampled states.

5 Conclusion

We revisited practical OPD by separating the sampler from the actor. This formulation exposes a simple choice: how much to optimize on the current batch before collecting another. Our fixed-batch experiments show that one update leaves substantial room to reduce disagreement with the teacher, motivating traj replay: take multiple actor steps on the same sampled states before refreshing the sampler. Across three student–teacher pairs, 128 updates per batch achieve comparable or better best-checkpoint accuracy without changing the loss, optimizer, or learning rate. At a matched budget of optimizer steps, this requires $128\times$ fewer generated and teacher-scored sequences and reduces FLOPs by 49%–64.9%. For Qwen3-1.7B in thinking mode, traj replay improves AVG_5 by 3.8 points over vanilla OPD while making training $2.5\times$ faster. Ablations show that the gains do not require an additional policy-KL penalty and are not reproduced by prompt replay, which keeps the sampler fixed but generates fresh trajectories. Together, these results support a simple principle: effective OPD depends not only on collecting new trajectories, but also on how thoroughly the student learns from the teacher on each batch.

AI USE STATEMENT

Generative AI tools were used in four ways. To aid and polish the writing: portions of the manuscript, including the abstract, were edited with AI assistance for clarity and readability. For retrieval and discovery: AI tools were used to find and screen related work. For research ideation and execution: AI tools assisted in developing the ideas, in writing and running the training, evaluation and plotting scripts, and in analyzing the results. To draft sections of the paper: parts of the text were first drafted with AI assistance. The authors reviewed and revised all AI-assisted text and code, verified every reported result, and take full responsibility for the content of the paper.

REPRODUCIBILITY STATEMENT

Our code is available at
https://anonymous.4open.science/r/optimize-more-sample-less-opd-FCFC/.
It contains the training stack behind every run in the paper, the launch scripts for the three student–teacher pairs, and the evaluation scripts. The method is specified by Algorithm 1 and the loss in Equation (3). Sections B and 4 give the models, prompt sets, and training and evaluation settings. Section A derives Equation (6), and Section C reports the per-benchmark accuracies behind the averaged curves.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2306.13649.

Charles Arnal, Vivien Cabannes, Taco Cohen, Julia Kempe, and Remi Munos. Efficient RL training for LLMs with experience replay, 2026. URL https://arxiv.org/abs/2604.08706.

Dami Choi, Alexandre Passos, Christopher J. Shallue, and George E. Dahl. Faster neural network training with data echoing. *arXiv preprint arXiv:1907.05550*, 2019. URL https://arxiv.org/abs/1907.05550.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Yuqian Fu, Haohuan Huang, Kaiwen Jiang, Jiacai Liu, Zhuo Jiang, Yuanheng Zhu, and Dongbin Zhao. Revisiting on-policy distillation: Empirical failure modes and simple fixes, 2026. URL https://arxiv.org/abs/2603.25562.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: On-policy distillation of large language models, 2026. URL https://arxiv.org/abs/2306.08543.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, et al. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report, 2025. URL https://arxiv.org/abs/2505.22312.

Qiangqiang He, Jin Li, and MingCai Chen. Calibrating teacher–student discrepancy for on-policy distillation, 2026. URL https://arxiv.org/abs/2609.21619.

Huipeng Huang and Hongxin Wei. Tail-aware top-k on-policy distillation, 2026. URL https://arxiv.org/abs/2608.14728.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjun Kang, Sanghyun Park, Donghoon Kim, Minjae Lee, Minseo Kim, Rishabh Tiwari, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. AsyncOPD: How stale can on-policy distillation be?, 2026. URL https://arxiv.org/abs/2606.24143.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe, 2026. URL https://arxiv.org/abs/2604.13016.

Kevin Lu and Thinking Machines Lab. On-policy distillation. Thinking Machines Lab: Connectionism, 2025. URL https://thinkingmachines.ai/blog/on-policy-distillation/.

Feng Luo, Yu-Neng Chuang, Guanchu Wang, Zicheng Xu, Xiaotian Han, Tianyi Zhang, and Vladimir Braverman. Demystifying OPD: Length inflation and stabilization strategies for large language models, 2026. URL https://arxiv.org/abs/2604.08527.

Miles Team. Miles: Enterprise-grade reinforcement learning for large-scale model post-training. https://github.com/radixark/miles, 2026.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3): 123–231, 2014.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

R. Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

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

Yecheng Wu, Song Han, and Hai Cai. Lightning opd: Efficient post-training for large reasoning models with offline on-policy distillation, 2026a. URL https://arxiv.org/abs/2604.13010.

Yecheng Wu, Song Han, and Han Cai. Lightning OPD 2.0: Mitigating style bias in cross-teacher on-policy distillation for large reasoning models, 2026b. URL https://arxiv.org/abs/2607.28449.

Wenda Xu, Rujun Han, Zifeng Wang, Long T. Le, Dhruv Madeka, Lei Li, William Yang Wang, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. Speculative knowledge distillation: Bridging the teacher–student gap through interleaved sampling. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.11325.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Dongxu Zhang, Zhichao Yang, Sepehr Janghorbani, Jun Han, II Ressler, Andrew, Qian Qian, Gregory D. Lyng, Sanjit Singh Batra, and Robert E. Tillman. Fast and effective on-policy distillation from reasoning prefixes, 2026a. URL https://arxiv.org/abs/2602.15260.

Yaocheng Zhang, Jiajun Chai, Yuqian Fu, Songjun Tu, Xiaohan Wang, Wei Lin, Guojun Yin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. Are full rollouts necessary for on-policy distillation?, 2026b. URL https://arxiv.org/abs/2605.31490.

A Derivation of the OPD gradient decomposition

We derive the two terms in Equation (6), treating φ and θ as independent parameters until after differentiation. For a given prompt x and trajectory y, the prefixes $s_t = (x, y_{<t})$ and length $|y|$ are fixed. Write $L(y;\theta) = \sum_{t=1}^{|y|} \ell(\theta; s_t)$ for the loss along this trajectory. Then $f(\varphi,\theta) = \mathbb{E}_{x\sim\mathcal{D}}\big[\sum_y \pi_\varphi(y \mid x) L(y;\theta)\big]$ is a weighted sum of trajectory losses.

Actor coordinate. When φ is held fixed, the weight $\pi_\varphi(y \mid x)$ of each trajectory does not depend on θ. Differentiation therefore acts only on $L(y;\theta)$, through the student's predictions at its fixed prefixes:

$$\begin{aligned}\partial_2 f(\varphi,\theta) &= \mathbb{E}_{x\sim\mathcal{D}}\Big[\sum_y \pi_\varphi(y \mid x)\nabla_\theta L(y;\theta)\Big] \\ &= \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\varphi(\cdot|x)}\Big[\sum_{t=1}^{|y|} \nabla_\theta \ell(\theta; s_t)\Big]. \end{aligned} \tag{8}$$

The second equality expands L into its local losses and writes the weighted sum over trajectories as an expectation under π_φ.

Sampler coordinate. When θ is held fixed, the loss $L(y;\theta)$ of each particular trajectory does not depend on φ. Changing φ changes only the probability assigned to that trajectory. We differentiate these probabilities and use $\nabla_\varphi \pi_\varphi(y \mid x) = \pi_\varphi(y \mid x)\nabla_\varphi \log \pi_\varphi(y \mid x)$:

$$\begin{aligned}\partial_1 f(\varphi,\theta) &= \mathbb{E}_{x\sim\mathcal{D}}\Big[\sum_y \nabla_\varphi \pi_\varphi(y \mid x) L(y;\theta)\Big] \\ &= \mathbb{E}_{x\sim\mathcal{D}}\Big[\sum_y \pi_\varphi(y \mid x)\nabla_\varphi \log \pi_\varphi(y \mid x) L(y;\theta)\Big] \\ &= \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\varphi(\cdot|x)}\Big[\nabla_\varphi \log \pi_\varphi(y \mid x) \sum_{t=1}^{|y|} \ell(\theta; s_t)\Big]. \end{aligned} \tag{9}$$

The last equality again rewrites the probability-weighted sum as an expectation and substitutes the definition of L.

Only after taking these partial derivatives do we set $\varphi = \theta$. The two expressions then give the first and second terms of Equation (6), respectively; their sum is $\nabla_\theta \mathcal{L}^{\text{ideal}}(\theta)$ by the chain rule for $\mathcal{L}^{\text{ideal}}(\theta) = f(\theta,\theta)$.

B Experimental details

Our training implementation builds on Miles (Miles Team, 2026). Tables 2 and 3 list the evaluation and training settings of the main runs; vanilla OPD and ours differ only in K.

Table 2: Evaluation settings of the three main runs.

	DeepSeek-R1-Distill-Qwen-1.5B	Qwen3-1.7B (thinking)	Qwen3-1.7B (non-thinking)
Samples per problem	32 (4 on MATH-500)	16 (4 on MATH-500)	16 (4 on MATH-500)
Benchmarks	AIME24, AIME25, AIME26, AMC23, MATH-500		
Response limit (tokens)	31,744		
Decoding	temperature 0.7, top-p 0.95		
Grader	boxed final answer, the training stack's own grader		
Evaluation frequency	every 16 optimizer steps		

Table 3: Training settings of the three main runs.

	DeepSeek-R1-Distill-Qwen-1.5B	Qwen3-1.7B (thinking)	Qwen3-1.7B (non-thinking)
Teacher	Skywork-OR1-Math-7B	Qwen3-4B-Thinking-2507	Qwen3-30B-A3B-Instruct-2507
Prompt set	OpenThoughts3-Math (fixed 224-prompt subset)	DAPO-Math-17k	DAPO-Math-17k
Thinking mode	on	on	off
Response limit (tokens)	16,384	16,384	8,192
Learning rate	10^{-5}	2×10^{-6}	5×10^{-6}
Rollout batch	112 prompts $\times$ 4 rollouts $=$ 448 sequences, one minibatch		
Sampling temperature	0.7		
Loss	teacher top-8 renormalized reverse KL (Equation (3)), no penalty		
Steps per batch K	128 (traj replay), 1 (vanilla OPD)		
Optimizer	Adam, $\beta = (0.9, 0.98)$, weight decay 0.1, constant learning rate		
Reported optimizer steps	256		
Random seed	1234		
Hardware	one node with eight NVIDIA B200 GPUs: seven for colocated student training and rollouts, one for the teacher server		

C PER-BENCHMARK RESULTS

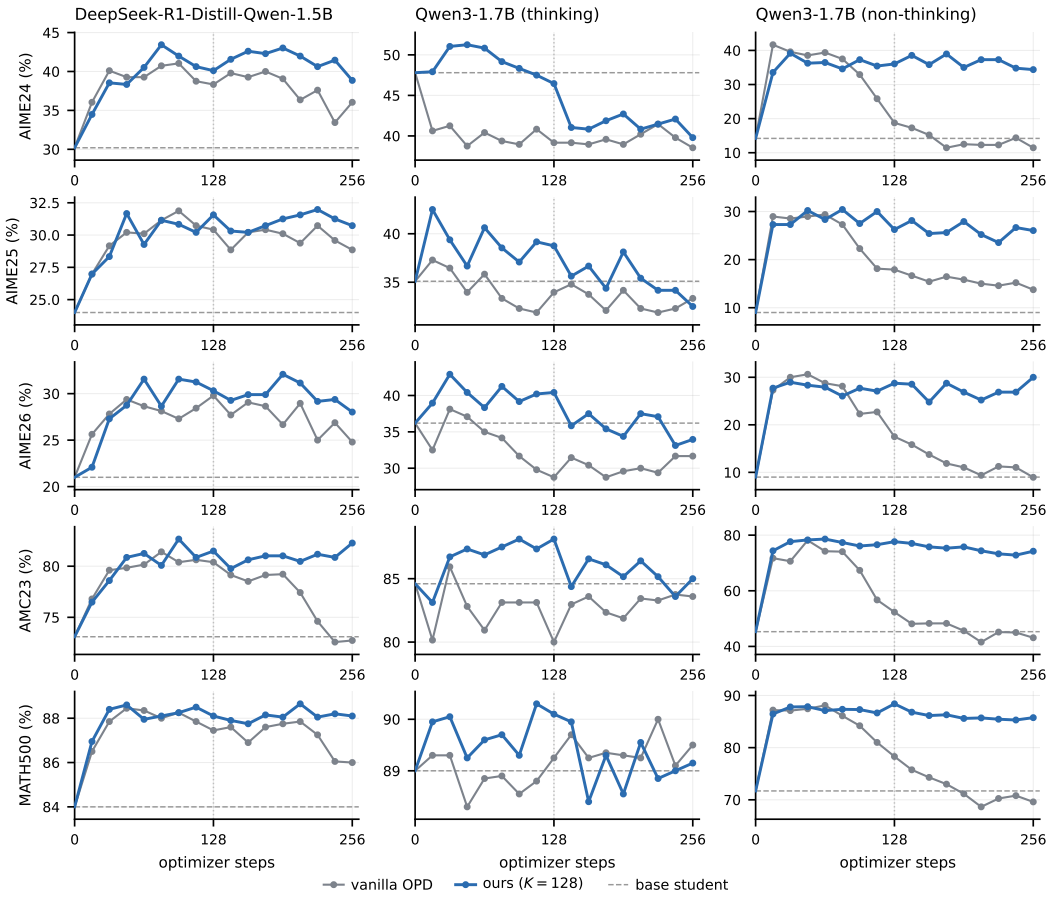

Figure 9: **Per-benchmark curves corresponding to Figure 4.** Columns: student–teacher pairs; rows: benchmarks. Dashed lines: base students; dotted lines: traj replay refresh at step 128.

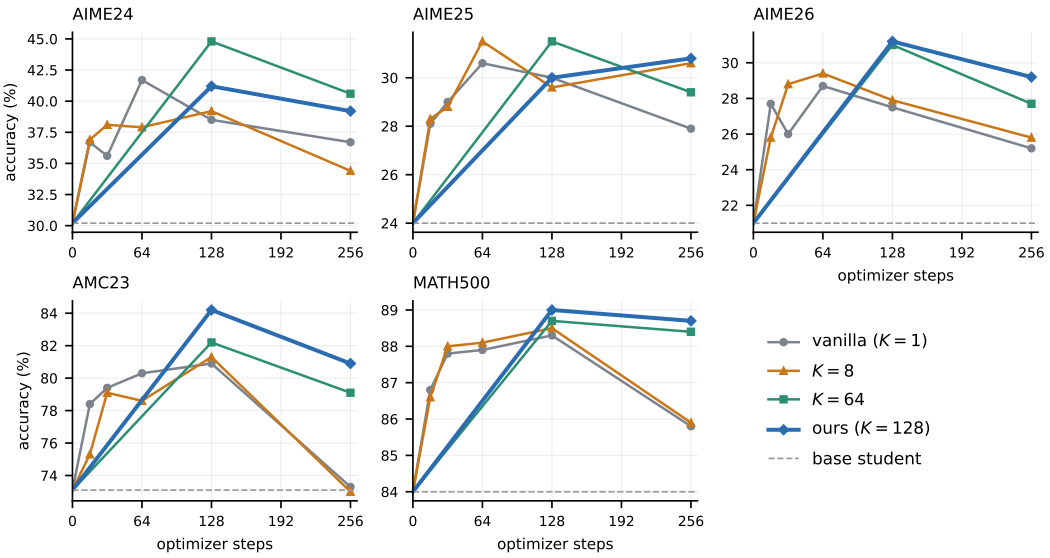

Figure 10: Per-benchmark accuracy against optimizer steps for the controlled sweep of $K \in \{1, 8, 64, 128\}$ in Figure 6a. Dashed line: the base student.

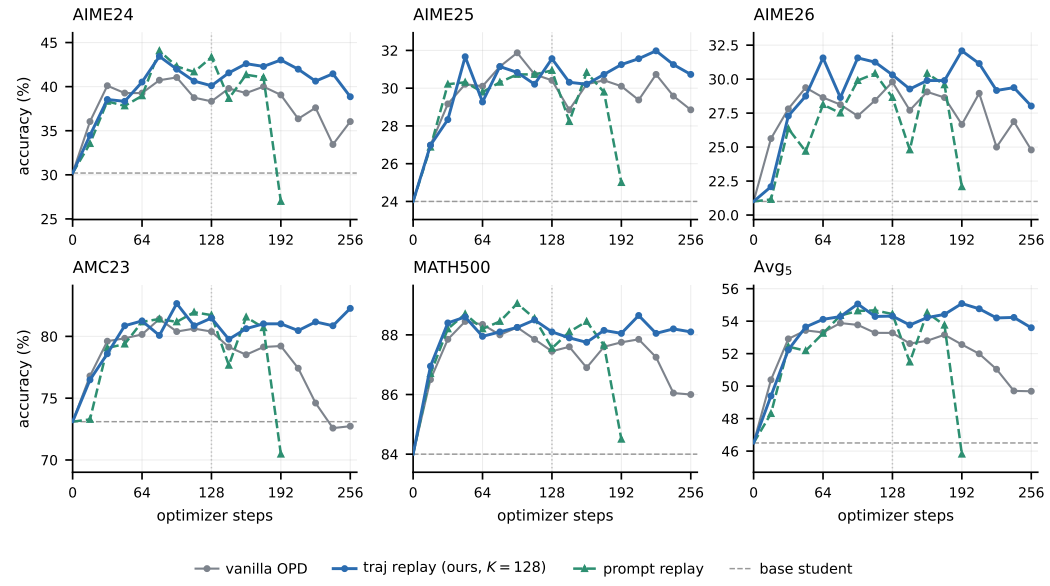

Figure 11: **Prompt replay versus traj replay, per benchmark.** DeepSeek-R1-Distill-Qwen-1.5B (Figure 8a). Prompt replay holds the sampling policy for 128 optimizer steps at a time and a fresh 448-sequence batch is drawn from it and teacher-scored at every step (dashed green; the base model for the first 128 optimizer steps, the step-128 student for the next 128; evaluated through step 192), against vanilla OPD and traj replay (ours, K=128) from the same lineage, per benchmark and on AVG$_5$. Dashed gray line: the base student; dotted line: the sampler refresh at step 128.