D^3-MOPD: Adaptive Dynamic Domain ScheDuling for Efficient Multi-Teacher Distillation

AllSpark Team

Multi-teacher on-policy distillation (MOPD) distills several domain-expert teachers into a single student by minimizing per-domain reverse-KL divergence on the student's own rollouts. Existing approaches typically fix the per-domain data mixture before training, overlooking the fact that different domains converge at substantially different rates: some plateau early while others continue to improve throughout the training budget. A fixed mixture therefore wastes compute on fast-converging domains and undertrains slower-converging ones. To address this, we propose D^3-MOPD (**D**ynamic **D**omain Sche**D**uling for MOPD), a zero-overhead scheduler that repurposes the per-domain reverse-KL signal already produced during training to adapt the domain mixture online. Running asynchronously outside the training process, an off-process watcher periodically tracks each domain's KL trajectory, estimates remaining headroom and current improvement rate, and accordingly adjusts the domain sampling ratios without altering the core training loop. Our D^3-MOPD scales naturally to arbitrary numbers of domains, and the expected benefit grows as more domains introduce more diverse convergence patterns for the scheduler to exploit. On a Qwen3.6-35B-A3B student distilled from four domain-expert teachers, D^3-MOPD closes 97% of the average student-to-teacher performance gap, compared with 63% for vanilla MOPD, reaches the same peak performance with an approximately 3× reduction in rollout steps, and surpasses the specialist teachers on three of seven benchmarks.

Date: August 22, 2026

AllSpark

1 Introduction

Multi-teacher on-policy distillation (MOPD) has recently emerged as an effective framework for combining the capabilities of several domain-expert teacher models into a single student [15, 4]. At each rollout step, the student generates on-policy trajectories from a K-domain prompt set [28, 31, 25], and each domain teacher prefills on the corresponding trajectories to produce its per-token distribution. The student then minimizes the per-token reverse-KL divergence [1, 7] between its own distribution and the teacher's along the trajectory. The result is a unified model that inherits the strengths of all teachers, without requiring all of them to be served at inference.

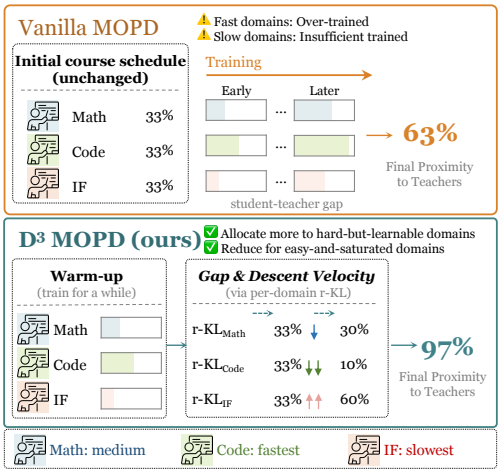

Figure 1: Comparison of vanilla MOPD and D^3-MOPD with $K = 3$ domains.

Current MOPD implementations typically rely on a static domain mixture $\{p_k\}$, fixed before training to match the prompt pool sizes. However, the student rarely learns each domain at the same rate [3, 16]: some domains plateau early as the student quickly closes the performance gap to the teacher, whereas others remain far from convergence throughout the entire training budget. Consequently, a fixed mixture wastes significant compute on converged domains while limiting the training budget for actively learning domains (Figure 1). Since the per-domain reverse-KL is already computed by the loss objective at every rollout step, a natural and promising solution is to repurpose it as a readily available signal for identifying and correcting this training imbalance.

In this paper, we propose ***D**ynamic **D**omain Sche**D**uling for MOPD* (D^3-MOPD), a lightweight scheduler that repurposes the per-domain reverse-KL signal already produced by the training loop to update the mixture $\{p_k\}$ online. An asynchronous watcher periodically converts each domain's KL history into a *composite sig-*

nal that combines its remaining gap to the teacher with its recent descent velocity, and maps this signal to a valid mixture via a temperature-controlled softmax with a per-domain floor. It then passes the updated mixture to the stratified data source, ensuring the core training loop remains strictly unmodified. By confining modifications to the data loader and running the watcher detached, D^3-MOPD remains compatible with advances in loss functions [27, 32, 10] and teacher training [24, 6], without additional training overhead.

We validate D^3-MOPD on a Qwen3.6-35B-A3B [21] student distilled from four domain-expert teachers spanning Math, Code, Instruction Following, and Tool-use. Experimental results highlight that D^3-MOPD outperforms vanilla MOPD, closing 97% of the average student-to-teacher performance gap (compared to 63% for the baseline) while achieving a higher peak accuracy across all tasks, surpassing the specialist teachers on three of seven benchmarks. It also improves training efficiency by reaching the baseline's optimal performance in just 47 rollout steps, approximately 3× faster than vanilla MOPD (143 steps). Furthermore, we argue that the expected benefits of D^3-MOPD will naturally scale with K, as more domains introduce more diverse convergence patterns for the scheduler to exploit. Overall, this paper first identifies a systematic mismatch between MOPD's static mixture and the student's asynchronous per-domain convergence, and resolves it with D^3-MOPD, a zero-overhead scheduler that turns the reverse-KL signal already computed by the training loop into a dynamic data mixture to improve both peak quality and training efficiency.

2 Motivation: Limitations of Fixed Domain Mixtures

In this section, we begin by formalizing vanilla MOPD and its fixed mixture (§2.1), then analyze per-domain learning dynamics in isolation (§2.2), and finally demonstrate how their disparate convergence rates waste compute under a static mixture (§2.3).

2.1 Vanilla MOPD and Fixed Mixtures

On-policy distillation (OPD) trains a student policy π_θ on its own generations, supervising them with a teacher π_T under a reverse Kullback-Leibler (KL) objective. Multi-teacher OPD (MOPD) extends this paradigm to K domains, where each domain is associated with a prompt distribution $\mathcal{D}_k$ and an expert teacher π_{T_k}. MOPD minimizes a weighted sum of per-domain losses:

$$\mathcal{J}_{\text{MOPD}}(\theta) \;=\; \sum_{k=1}^{K} p_k \, \mathbb{E}_{x\sim\mathcal{D}_k,\, y\sim\pi_\theta(\cdot|x)} \Big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid x,y) \,\|\, \pi_{T_k}(\cdot \mid x,y)\big)\Big], \tag{1}$$

where the KL divergence is summed over the generated tokens of y, and the mixture weight p_k satisfies $p_k \geq 0$ with $\sum_k p_k = 1$. In vanilla MOPD, the per-domain datasets $\mathcal{D}_k$ are pooled and shuffled, resulting in an expected per-batch share for domain k of $p_k = N_k / \sum_i N_i$, where $N_k = |\mathcal{D}_k|$. Crucially, this expected share remains strictly defined prior to training and is held constant throughout the run. Pool-and-shuffle sampling fixes only this expectation, and each batch's realized share fluctuates around p_k.

2.2 Domain-Specific Learning Dynamics

To understand the inefficiency of a static p_k, we analyze each domain. Specifically, we train the student separately on Math, Code, and Instruction Following (IF) using single-domain OPD, with identical initializations and the corresponding expert teacher for each domain. Figure 2 reports the per-step reverse-KL on a held-out evaluation set. Math drops sharply during the first quarter of the training budget before plateauing near a floor. Code declines steadily at a slower rate, showing no signs of convergence within the budget. Meanwhile, IF continues to reduce its loss throughout, yet its absolute reverse-KL remains one to two orders of magnitude above the other domains at every step. In conclusion, these domains reach their plateau phases at different stages, with their absolute KL signals varying by orders of magnitude.

2.3 Computational Inefficiency of Fixed Mixtures

When these same three domains are trained jointly under vanilla MOPD with a uniform mixture ($p_{\text{math}} = p_{\text{code}} = p_{\text{if}} = 1/3$), the varying convergence rates documented in §2.2 translate into a substantial waste of training compute. As illustrated in Figure 3, the three domains enter their low-KL phases at markedly different points: Code by roughly step 48, Math by step 96, and IF only around step 144. Yet the fixed mixture allocates one-third of every batch to each domain for the remainder of training. Consequently,

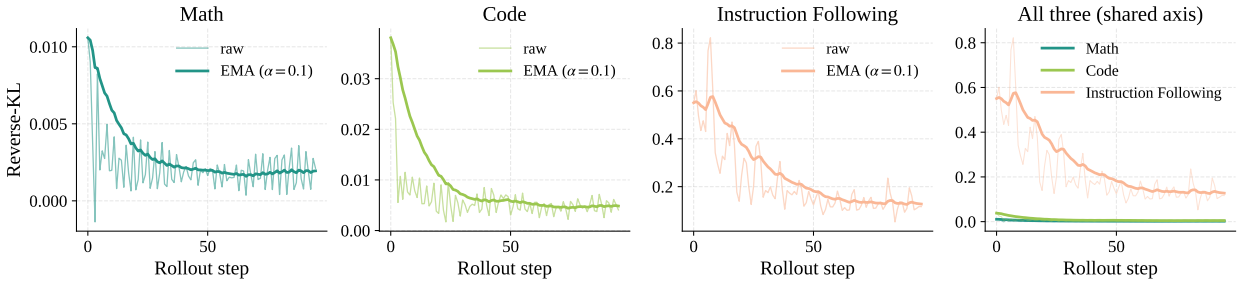

Figure 2: Per-domain OPD reverse-KL on Math, Code, and IF. The three domains converge at widely different rates, with the absolute KL of IF remaining one to two orders of magnitude above the others throughout the training process.

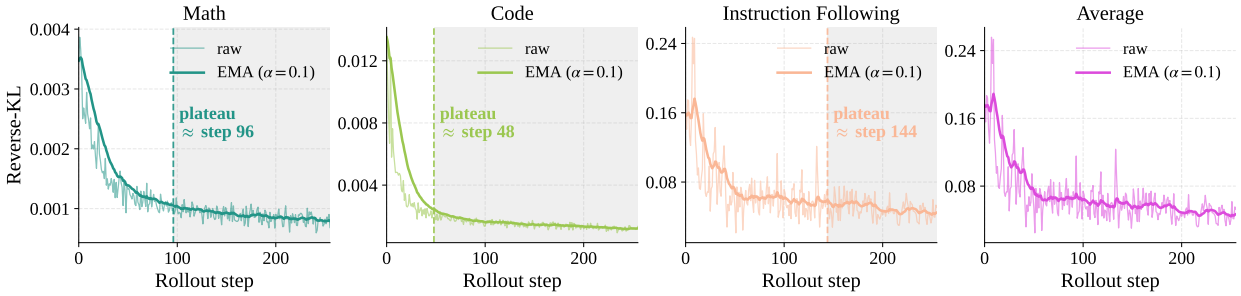

Figure 3: Vanilla MOPD with a fixed uniform mixture ($p_k = 1/3$). The three domains reach their low-KL phases at markedly different points, yet the static mixture allocates one-third of every batch to each throughout the remaining training (shaded regions).

valuable training compute is wasted on domains long past the point where additional samples yield further KL reductions. This waste stems not from the data or the teachers themselves, but from holding p_k fixed.

Takeaway. Under vanilla MOPD, distinct domains enter low-KL plateau phases at entirely different training stages (§2.2). A fixed mixture inevitably wastes compute on domains that have ceased to improve (§2.3). This highlights the necessity for a dynamic sampling ratio p_k that smoothly adapts to native training signals, motivating the design of our method in §3.

3 Method

Based on these observations, we propose **D**ynamic **D**omain Sche**D**uling for MOPD (D^3-MOPD), a lightweight scheduler that adapts the per-domain sampling ratio $\{p_k\}$ using the reverse-KL signal. §3.1 describes the framework, which consists of an off-process watcher and a stratified data source. §3.2 then details how the per-domain KL history is converted into a dynamic mixture.

3.1 Framework

D^3-MOPD introduces two components to a vanilla MOPD training loop, as illustrated in Figure 4: an *off-process watcher* that computes the per-domain mixture p_k from the reverse-KL signals already produced by the trainer, and a *stratified data source* that reads p_k to assemble each batch with the target mixture. This pipeline generalizes to any number of domains K.

The trainer follows vanilla MOPD: at each rollout step, the student generates responses from prompts drawn by the data source. Each response is dispatched to its corresponding domain teacher, which prefills on the trajectory to produce its per-token distribution. The resulting per-domain reverse-KL is appended to a shared log. Since vanilla MOPD typically logs only the domain-averaged reverse-KL, the watcher groups per-sample KL values by domain to recover per-domain signals. It runs as a separate process that periodically computes a new mixture p_k following §3.2 and writes it to a status file that the data source reads between updates so that the trainer never blocks.

We replace the conventional pool-and-shuffle loader with the *stratified data source*. Given a batch size B and per-batch mixture $\{\tilde{p}_k\}$ (obtained by applying the batch-level jitter below to the watcher-supplied $\{p_k\}$),

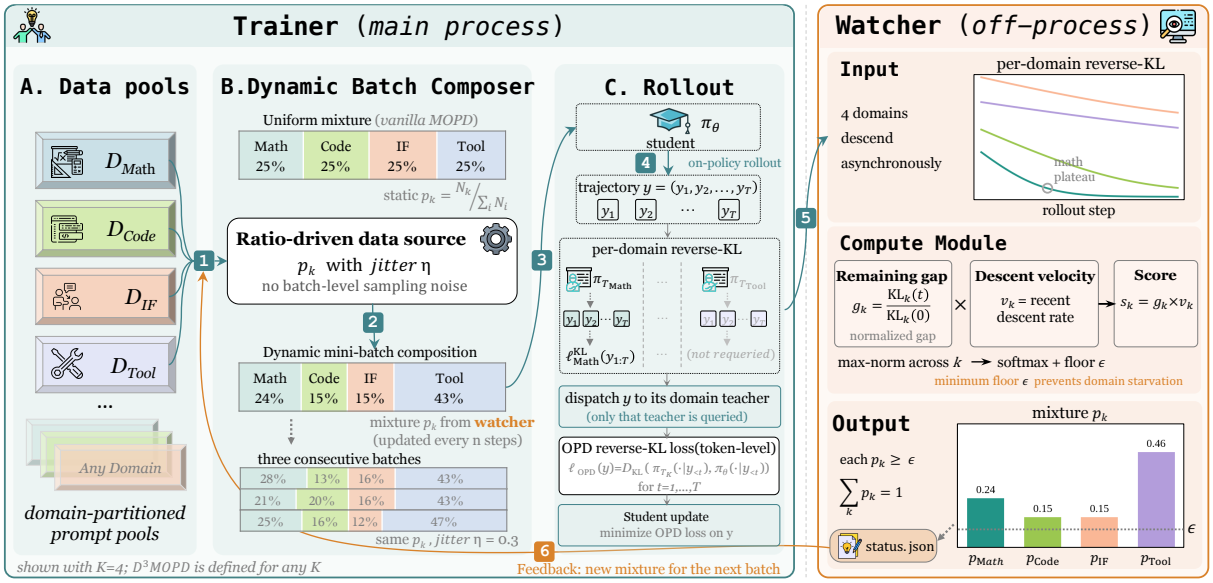

Figure 4: The framework of D^3-MOPD. At each rollout step, the student generates responses that the teachers prefill on to produce per-domain reverse-KL signals. A watcher periodically maps these signals into a continuous per-domain mixture p_k that shapes subsequent batches.

it first assigns $\lfloor B \cdot \tilde{p}_k \rfloor$ samples to each domain, then distributes the remaining $B - \sum_k \lfloor B \cdot \tilde{p}_k \rfloor$ samples to the domains with the largest fractional parts. This strictly aligns each domain's per-batch share with the target mixture, replacing the uncontrolled variance of pooled shuffling with the purposeful batch-level jitter described below.

Batch-level jitter. Because the watcher updates the mixture only every n rollout steps, consecutive batches would otherwise share identical domain proportions. The data source therefore perturbs each batch's proportions via $\tilde{p}_k = p_k(1+u_k)/\sum_j p_j(1+u_j)$ with $u_k \sim \text{Uniform}(-\eta, \eta)$, restoring batch-to-batch variation while keeping the long-run mean close to $\{p_k\}$ ($\mathbb{E}[u_k] = 0$).

Integration with existing MOPD. All modifications are strictly confined to the data path, leaving the rollout, teacher prefill, and student update operations untouched. An existing MOPD implementation can adopt D^3-MOPD by replacing the standard loader with the stratified data source and launching a watcher process alongside the trainer.

3.2 Composite Ratio from Remaining Gap and Descent Velocity

The watcher translates the per-domain reverse-KL history into a mixture $\{p_k\}$ at each update. Our goal is to allocate more samples to domains exhibiting ongoing improvement and fewer to those that have converged. We quantify this improvement along two axes: the *remaining gap* $\widetilde{\text{KL}}_k$ (the proportion of the initial KL left to close) and the *descent velocity* v_k (the recent rate of KL reduction). Relying on either axis alone is insufficient. A domain might exhibit a large remaining gap but plateau, rendering further samples ineffective. Conversely, a nearly converged domain might descend rapidly within the current window, yet its minimal room for improvement makes additional samples redundant. Their product ensures that a domain receives a large share only when it has room to improve and is actively doing so.

Remaining gap. We define the remaining gap by normalizing the current smoothed KL by its initial value,

$$\widetilde{\text{KL}}_k(t) = \frac{\overline{\text{KL}}_k(t)}{\text{KL}_k^{(0)}}, \qquad \text{KL}_k^{(0)} = \frac{1}{S_0}\sum_{t=1}^{S_0} \text{KL}_k(t), \tag{2}$$

where $\overline{\text{KL}}_k(t)$ is the exponential moving average (EMA) of domain-k reverse-KL up to rollout step t, and $\text{KL}_k^{(0)}$ is the mean of the first S_0 observations seeded once at the start of training. Under normal training,

$\widetilde{\mathrm{KL}}_k(t) \in (0, 1]$ and approaches zero as the domain converges. As documented in §2.2, this normalization enables fair comparison across domains whose absolute KL values may differ by orders of magnitude.

Descent velocity. To determine whether a domain's KL continues to decrease, we measure its recent rate of change. Let W be a fixed window length in rollout steps and R the number of non-overlapping windows used for averaging. We define the single-window relative change as

$$\delta_k^{(i)}(t) \;=\; \frac{\overline{\mathrm{KL}}_k(t - iW) - \overline{\mathrm{KL}}_k(t - (i+1)W)}{\overline{\mathrm{KL}}_k(t - (i+1)W)}, \qquad i = 0, \ldots, R-1, \tag{3}$$

where $\delta_k^{(0)}$ represents the most recent window. The descent velocity takes the negative average of these R changes and clips the result at zero,

$$v_k(t) \;=\; \max\Big(0,\; -\frac{1}{R}\sum_{i=0}^{R-1} \delta_k^{(i)}(t)\Big), \tag{4}$$

meaning $v_k(t) > 0$ strictly requires the domain's KL to decrease on average over the last R windows.

Composite signal. We compute the composite signal via the product of these two metrics,

$$s_k(t) \;=\; \widetilde{\mathrm{KL}}_k(t) \cdot v_k(t), \qquad \tilde{s}_k(t) \;=\; s_k(t) \;/\; \max_j s_j(t), \tag{5}$$

ensuring s_k remains large only when a domain is both far from convergence and steadily descending. The max-normalization brings $\tilde{s}_k$ into $[0, 1]$, making the softmax temperature T independent of the raw signal magnitude. In the degenerate case where all domains plateau and $\max_j s_j(t) = 0$, we set $\tilde{s}_k(t) = 0$ for all k, and the mapping defaults to a uniform mixture. Appendix E provides a theoretical analysis showing that under an exponential decay model, s_k approximates the normalized marginal KL reduction, justifying the gap-velocity product as a greedy allocation rule.

Mapping to a valid mixture. We map $\tilde{s}_k$ to a probability mixture through a softmax function parameterized by temperature T and a per-domain floor ϵ,

$$p_k(t) \;=\; \epsilon \;+\; \big(1 - K\epsilon\big) \cdot \frac{\exp\big(\tilde{s}_k(t)/T\big)}{\sum_{j=1}^{K} \exp\big(\tilde{s}_j(t)/T\big)}, \tag{6}$$

guaranteeing that $p_k(t) \geq \epsilon$ for every domain and $\sum_k p_k(t) = 1$. The watcher recomputes this mixture at every update and delivers it to the data source via the status file described in §3.1.

Why average over R windows. With $R = 1$, the velocity estimate is highly sensitive to random noise in the KL signal. Because Eq. 4 clips the sum at zero, a short-term KL spike in an actively learning domain would wrongly push v_k to zero, cutting off its sample allocation. Averaging the changes over R non-overlapping windows prevents these brief spikes from accidentally triggering the cutoff. As a result, v_k becomes zero only when the domain remains flat or rises over a longer period, making it a reliable sign of actual convergence.

Warmup. The composite signal becomes available once $2W$ observations of $\overline{\mathrm{KL}}_k$ are recorded. At this point, $\widetilde{\mathrm{KL}}_k$ relies on the seeded $\mathrm{KL}_k^{(0)}$, and computing a single $\delta_k^{(0)}$ requires the two window endpoints at $t-W$ and t. Eq. 4 represents the standard formulation with a full history of R windows. During the warmup phase $2W \leq t < (R+1)W$, the algorithm replaces R with the available window count $R'(t) = \lfloor t/W \rfloor - 1$. From $t = (R+1)W$ onward, $R'(t)$ is capped at R, and the standard form of Eq. 4 takes over. Prior to $t = 2W$, the watcher provides no mixture updates, causing the data source to default to a uniform mixture $p_k = 1/K$. The batch-level jitter (§3.1) remains active throughout this initial period.

Design rationale. The temperature T modulates how sharply the sampling share is redirected: as $T \to \infty$, the mixture becomes uniform, whereas as $T \to 0$, the mixture concentrates on the argmax of $\tilde{s}$. Finally, the floor ϵ prevents any domain from being completely discarded, ensuring already-converged domains retain a minimal presence to mitigate catastrophic forgetting. Algorithm 1 summarizes the scheduling process.

Algorithm 1 **D**ynamic **D**omain Sche**D**uling for MOPD (D^3-MOPD).

Require: Domain pools $\{\mathcal{D}_k\}_{k=1}^{K}$; hyperparameters $T, \epsilon, S_0, R, W, n, \eta, B$; shared KL log and status file.

Initialize at step $t = S_0$: $\mathrm{KL}_k^{(0)} \leftarrow \frac{1}{S_0}\sum_{\tau=1}^{S_0}\mathrm{KL}_k(\tau)$ for all k ▷ Eq. 2; seeded once
function WATCHERTICK(t) ▷ invoked at rollout step t, every n steps
  if $t < 2W$ **then return** ▷ initial warmup; data source uses uniform fallback
  end if
  $R' \leftarrow \min\big(R,\ \lfloor t/W \rfloor - 1\big)$ ▷ $R' = R$ once $t \geq (R{+}1)W$
  for $k = 1, \ldots, K$ **do**
    $\widetilde{\mathrm{KL}}_k \leftarrow \overline{\mathrm{KL}}_k(t)/\mathrm{KL}_k^{(0)}$ ▷ Eq. 2
    $\delta_k^{(i)} \leftarrow \big[\overline{\mathrm{KL}}_k(t{-}iW) - \overline{\mathrm{KL}}_k(t{-}(i{+}1)W)\big]/\overline{\mathrm{KL}}_k(t{-}(i{+}1)W)$ for $i = 0, \ldots, R'{-}1$
    $v_k \leftarrow \max\big(0,\ -\frac{1}{R'}\sum_i \delta_k^{(i)}\big)$ ▷ Eq. 4
    $s_k \leftarrow \widetilde{\mathrm{KL}}_k \cdot v_k$
  end for
  if $\max_j s_j = 0$ **then** ▷ degenerate case: all domains plateaued
    $\tilde{s}_k \leftarrow 0$ for all k
  else
    $\tilde{s}_k \leftarrow s_k / \max_j s_j$ for all k ▷ Eq. 5
  end if
  $p_k \leftarrow \epsilon + (1{-}K\epsilon)\,\dfrac{\exp(\tilde{s}_k/T)}{\sum_j \exp(\tilde{s}_j/T)}$ for all k ▷ Eq. 6
  Atomically write $\{p_k\}$ to the status file
end function
function GETBATCH(B) ▷ invoked at every rollout step
  $\{p_k\} \leftarrow$ status file (fallback $p_k \leftarrow 1/K$)
  $u_k \sim \mathrm{Uniform}(-\eta, \eta)$; $\tilde{p}_k \leftarrow p_k(1{+}u_k)/\sum_j p_j(1{+}u_j)$ ▷ jitter + renormalize
  Assign $n_k \leftarrow \lfloor B\tilde{p}_k \rfloor$; distribute the remaining $B - \sum_k n_k$ samples to the domains with the largest fractional parts so $\sum_k n_k = B$
  return n_k prompts sampled from $\mathcal{D}_k$ for each k
end function

4 Experiments

4.1 Training Setup

We validate D^3-MOPD using Qwen3.6-35B-A3B [21] as the student model and K=4 domain-expert teachers that share the same backbone but are individually fine-tuned via GRPO [22] on a single domain: Math, Code, Instruction Following, or Tool-use, extending the three-domain analysis of §2 with a fourth domain. We optimize the student using the vanilla MOPD [16] per-token reverse-KL (r-KL) objective on the slime [35] framework, which employs an asynchronous rollout-update loop that overlaps teacher prefill with student generation. The composite signal watcher operates as an off-process job alongside the trainer following §3.1. It recomputes the mixture $\{p_k\}$ every $n = 10$ rollout steps and delivers it to the stratified data source via a status file. All main runs use a rollout batch size of 128 over 256 rollout steps. Other training details are provided in Appendix B.

Training data. We construct a prompt set of ∼4k prompts per domain, formatted as single-turn message lists in JSONL with per-sample domain tags. The Math and Code splits come from DAPO-Math-17K [30] and CodeI/O [12], respectively. The Instruction Following split is filtered from Nemotron-Cascade 2 [28], and the Tool-use split uses the Fission-GRPO [33] training set.

4.2 Evaluation Setup

We evaluate on seven benchmarks covering the four training domains: AIME 2025 [17] (avg@64) and HMMT November 2025 [5] (avg@32) for mathematics; LiveCodeBench Code Generation [9] (avg@6) and OJBench C++ [26] (default protocol) for code; IFBench [19] and IFEval [34] for instruction following; and BFCL v3 Multi-Turn (Base) [18] for tool-use. We generate responses in non-thinking mode with temperature 0.7, top-p 0.8, top-k 20, and presence penalty 1.5, following the official Qwen3.6-35B-A3B recommendations.

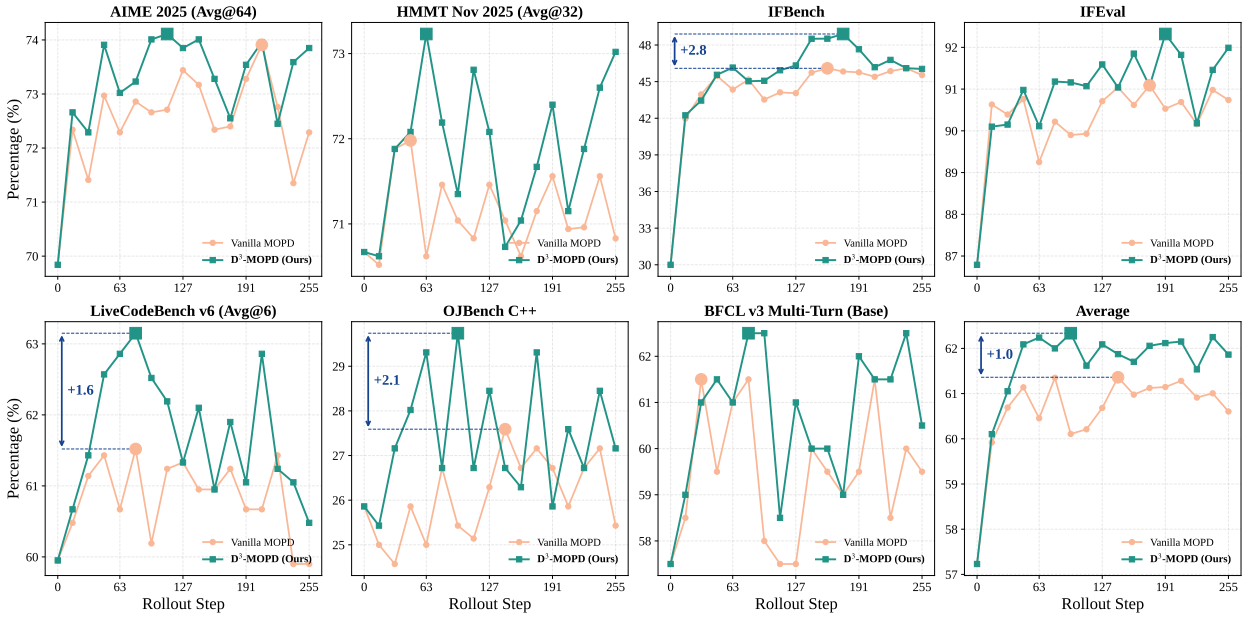

Figure 5: Per-benchmark accuracy of vanilla MOPD and D^3-MOPD across the 16 evaluated rollout checkpoints, spanning seven benchmarks plus their unweighted average. D^3-MOPD matches or outperforms the vanilla baseline on every benchmark at almost every checkpoint.

We evaluate every 16 rollout steps starting from step 15. D^3-MOPD and vanilla MOPD are trained under identical settings except for the mixture policy: fixed $\{p_k\}$ vs. online-updated $\{p_k\}$. Since raw student-to-teacher performance gaps differ across domains, averaging standard accuracies over-weights domains with larger margins. We therefore report a per-benchmark normalized score $\hat{s}_b = (s_b - s_b^{\text{stu}})/(s_b^{\text{tea}} - s_b^{\text{stu}})$, which scales the initial student to 0 and the domain-expert teacher to 1, and compute its uniform average across benchmarks. Values above 1.0 indicate improvement beyond the respective specialist teacher.

4.3 Main Results

Figure 5 shows the per-benchmark accuracy of D^3-MOPD and vanilla MOPD over the training budget. As shown in the average panel, both methods improve rapidly during the first $\sim$50 steps before plateauing, with D^3-MOPD consistently maintaining a higher overall trajectory. The composite signal first activates at step 20 (remaining-gap component only) and integrates the velocity component from step 40. Consequently, the initial evaluation at step 15 shows minimal performance differences. Overall, D^3-MOPD outperforms vanilla MOPD in both peak performance and training efficiency. We highlight three primary findings:

- **D^3-MOPD peaks higher on all benchmarks.** The vanilla baseline reaches its optimal average of 61.4 at step 143. In contrast, D^3-MOPD attains a higher peak of 62.3 as early as step 95, demonstrating individual benchmark gains ranging from $+0.2$ on AIME 2025 (74.1 vs 73.9) to $+2.8$ on IFBench (48.9 vs 46.1). The early peak in average accuracy occurs because code benchmarks (LiveCodeBench and OJBench) converge and begin degrading after $\sim$80 steps, consistent with the domain over-optimization observed in §2 (Figure 3). By reducing the share of these converging domains, the scheduler redirects the rollout budget toward domains with active growth, raising the overall peak accuracy.
- **D^3-MOPD reaches the vanilla peak with $3\times$ fewer steps.** While vanilla MOPD requires 143 steps to reach its average score of 61.4, D^3-MOPD surpasses this threshold by step 47 (scoring 62.1). Furthermore, vanilla MOPD fails to establish its per-benchmark peaks until as late as step 207 (AIME 2025). Notably, D^3-MOPD matches or exceeds every one of these individual peaks within the first 79 steps. Because the scheduler continuously shifts prompt sampling from converging domains to those still improving, each rollout step yields a larger marginal gain, accelerating overall convergence.
- **Domain convergence order aligns with r-KL.** Code benchmarks peak earliest (LiveCodeBench by step 79, OJBench by step 95), whereas instruction-following benchmarks peak latest (IFBench at step 175,

Table 1: Ablation results. Best-S denotes the rollout step whose checkpoint achieves the highest average across the evaluated benchmarks over the full 256-step run. All variants differ from D^3-MOPD in only the ablated variable; other settings are identical.

		Math		IF		Code		Tool	
Method	Best-S	AIME25	HMMT(N)	IFBench	IFEval	LCB v6	OJB C++	BFCL(B)	Avg
Vanilla MOPD	143	73.17	71.04	45.74	91.04	60.95	27.59	60.00	61.36
w/o velocity	127	72.66	71.15	45.17	90.91	59.52	28.45	62.00	61.41 (+0.05)
w/o gap	255	71.72	**71.46**	**49.01**	90.83	60.10	24.57	62.50	61.46 (+0.10)
w/o jitter (η=0)	95	72.66	71.15	46.58	89.90	60.95	27.16	63.00	61.63 (+0.27)
w/o smoothing (R=1)	95	73.07	**71.46**	46.94	**92.14**	61.81	26.29	**63.50**	62.17 (+0.81)
D^3-MOPD (η=0.30, R=3)	95	**74.01**	71.35	45.07	91.16	**62.52**	**29.74**	62.50	**62.34** (+0.98)

IFEval at step 191), while mathematics falls in between. This progression aligns with the r-KL convergence pattern in Figure 3, where code r-KL plateaus first and IF r-KL last. This consistent alignment validates r-KL as an effective monitoring signal: a domain with a plateaued r-KL has likely approached its peak accuracy, and further budget allocation risks computational waste and model degradation.

4.4 Ablation Studies

Table 1 evaluates the individual contributions of three design choices in D^3-MOPD. The per-domain ratio trajectories of each variant are provided in Appendix G.

Composite signal vs. single-signal variants. Although both single-signal variants outperform vanilla MOPD (w/o velocity 61.41, w/o gap 61.46) and validate r-KL-driven dynamic scheduling, neither matches the composite formulation (62.34). The two signals differ in convergence speed. The w/o velocity variant peaks early at step 127 because it uses the current normalized r-KL levels to identify plateauing domains. In contrast, the w/o gap variant peaks at step 255. Since all domains' KL drops rapidly during early training, the velocity signal cannot differentiate them until descent rates diverge. Notably, the w/o gap variant records the highest IFBench score (49.01), consistent with IF converging last (§4.3). A velocity-driven scheduler allocates budget to IF since its descent rate remains high, enabling prolonged capability growth.

Effectiveness of batch jitter. Removing jitter (η=0) does not alter the peak step (both at 95) but reduces the peak average by 0.71 (61.63 vs 62.34), with the most severe drops on code benchmarks (OJBench −2.6, LiveCodeBench −1.6). We hypothesize that the proportion variation from jitter acts as a form of exploration: for domains nearing their r-KL plateau, a fixed sampling ratio yields only weak gradients, and the randomized bursts from jitter may deliver stronger updates. This would explain why the rapidly plateauing code benchmarks benefit the most. The identical peak step confirms that jitter improves overall training quality without altering convergence speed.

Effectiveness of velocity smoothing. The hyperparameter R controls how many non-overlapping observation windows are averaged to estimate descent velocity: R=1 relies on a single window (reactive but noisy), whereas D^3-MOPD adopts R=3 for smoother estimates. Both variants peak at step 95, and R=1 already achieves 62.17 (+0.81 over vanilla), indicating that even a single-window velocity captures useful convergence information. R=3 further improves the average to 62.34 by mitigating random r-KL variance. This smoothing prevents a single noise-induced r-KL fluctuation from temporarily zeroing the velocity signal and cutting off budget to actively learning domains. Interestingly, R=1 surpasses D^3-MOPD on BFCL (63.50 vs 62.50), likely because tool-use exhibits high cross-checkpoint variance, where a noisier estimate encourages beneficial exploration.

4.5 Supplementary Analysis

D^3-MOPD closes 97% of the student-teacher gap. Table 2 compares peak accuracy against the student baseline and domain-expert teachers. Vanilla MOPD closes 63% of the average student-to-teacher gap ($\bar{s}$ =

Table 2: Per-benchmark peak accuracy (%) and normalized score (§4.2) over 256 rollout steps. Each value represents the maximum across all 16 checkpoints. Δ rows show gaps to the domain-expert teacher. Bold marks the column best, underline the runner-up.

	Math		Instruction Following		Code		Tool-use	
Method	AIME25	HMMT(N)	IFBench	IFEval	LCB-v6	OJB-C++	BFCL(B)	Norm.
Teacher	**76.2**	<u>72.3</u>	**52.1**	<u>91.9</u>	**64.8**	<u>28.5</u>	**67.0**	**1.00**
Student	69.8	70.7	30.0	86.8	60.0	25.9	57.5	0.00
Δ(Student − Teacher)	−6.4	−1.6	−22.1	−5.1	−4.8	−2.6	−9.5	−1.00
Vanilla MOPD	73.9	72.0	46.1	91.1	61.5	27.6	61.5	0.63
Δ(Vanilla − Teacher)	−2.3	−0.3	−6.0	−0.8	−3.3	−0.9	−5.5	−0.37
D^3-MOPD (Ours)	<u>74.1</u>	**73.2**	<u>48.9</u>	**92.3**	<u>63.2</u>	**29.7**	<u>62.5</u>	<u>0.97</u>
Δ(D^3-MOPD − Teacher)	−2.1	+0.9	−3.2	+0.4	−1.6	+1.2	−4.5	−0.03

0.63), whereas D^3-MOPD closes 97% of this gap ($\bar{s} = 0.97$). Consistent with findings by Ma et al. [16], both methods generally remain below the maximum performance of the domain-expert teachers, indicating that distillation primarily closes existing capability gaps rather than generating novel skills. Notably, D^3-MOPD surpasses the teacher on three benchmarks (HMMT(N), IFEval, and OJBench C++, $\hat{s}_b > 1.0$), whereas vanilla MOPD remains below the teacher on all tasks. Because these three benchmarks exhibit small initial student-to-teacher absolute accuracy gaps ($\Delta \leq 5.1$), even modest absolute improvements translate into normalized scores exceeding 1.0. Appendix C further compares the two methods at their respective best-average checkpoints, confirming that D^3-MOPD's improvement persists under this setting.

D^3-MOPD shifts budget from converging to active domains. Figure 6 shows the per-domain sampling ratio throughout training. While vanilla MOPD maintains all four domains near 0.25 with only batch-level noise, D^3-MOPD dynamically adapts this allocation, closely tracking domain convergence. Code converges fastest (its r-KL plateaus first in Figure 3) and is steadily downsampled from 0.25 to ∼0.15 throughout training. The freed budget flows initially to Math, whose ratio rises to ∼0.50 during steps 60–127 as the composite signal identifies Math as the domain with the largest remaining gap and active descent. Once Math converges after step 127, its ratio recedes, and the scheduler redirects budget toward Instruction Following and Tool-use, pushing their ratios to ∼0.55 and ∼0.50 after step 150. Because these two domains exhibit the slowest r-KL convergence, this late-stage budget increase enables the student to sustain progress where capacity for improvement remains high. Ultimately, this dynamic reallocation functions as an implicit curriculum: each domain receives concentrated training during the window when it can still improve, rather than having the budget split uniformly regardless of learning state.

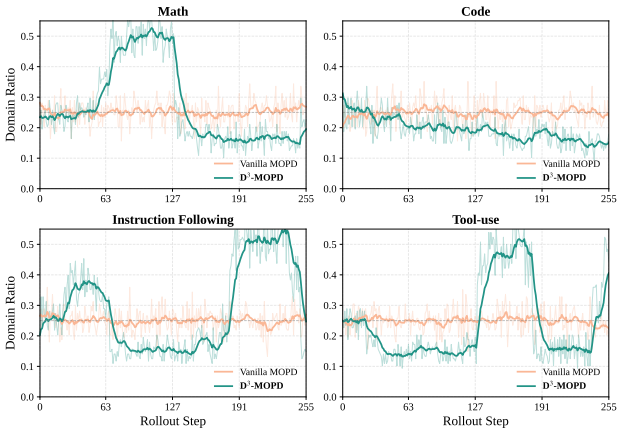

Figure 6: Per-domain sampling ratio of vanilla MOPD and D^3-MOPD. D^3-MOPD dynamically reallocates budget as domains converge.

5 Related Work

5.1 On-Policy Distillation

On-policy distillation (OPD) trains a student on its own generations, supervised by the teacher's distribution under a reverse Kullback-Leibler objective [7, 1]. Recent work refines this supervision at the token or sample level: Jin et al. [11] switch between forward and reverse KL based on the teacher's entropy, Li et al. [14] filter and reweight tokens to suppress noisy gradients, and Hou et al. [8] correct the mismatch between per-token KL and sequence-level outcomes. Other directions stabilize the learning target [10], distill selectively from reasoning prefixes [32], or extend beyond the teacher via reward signals [27]. Li et al.

[13] provide a systematic analysis of these design choices. These methods operate on the local supervision signal, improving which tokens or samples the student learns from. In this paper, we address a complementary dimension: instead of refining per-token supervision, we dynamically adjust the per-domain data allocation when multiple teachers are involved.

5.2 Multi-Teacher On-Policy Distillation

Multi-teacher OPD (MOPD) extends the single-teacher setting by distilling several domain-expert teachers into one student [16]. Recent large-scale post-training pipelines adopt MOPD-style distillation: MiMo-V2-Flash [15] and DeepSeek-V4 [4] use it for capability integration, while Baichuan-M3 [25], Nemotron-Cascade 2 [28], and GLM-5 [31] apply it in vertical-domain or staged pipelines. Chen et al. [3] further observe that domain teachers can pull the student in conflicting directions, and mitigate this through alternating per-domain training; Shen et al. [23] and Yin et al. [29] address failure modes such as tool-call boundary drift and heterogeneous teacher confidence. However, all existing methods keep the per-domain data ratio fixed throughout training and therefore cannot react to the asynchronous convergence documented in §2. In this paper, we revisit MOPD from a scheduling perspective: the per-domain data ratio is treated as a continuous variable, updated online from the reverse-KL signal the training loop already produces.

6 Conclusion

This paper identifies and addresses a systematic inefficiency in multi-teacher on-policy distillation (MOPD) arising from the mismatch between its static domain mixture and the student's asynchronous per-domain convergence rates. D^3-MOPD resolves this by combining each domain's remaining KL gap with its descent velocity into a composite signal and mapping the result to an updated sampling ratio through a softmax-floor formulation, all without modifying the core training loop. Experiments across four domains with a Qwen3.6-35B-A3B student show that D^3-MOPD closes 97% of the student-teacher performance gap (vs. 63% for vanilla MOPD) while reaching the baseline's peak accuracy in $3\times$ fewer rollout steps. Ablation studies confirm that each design choice, namely the composite signal, batch jitter, and velocity smoothing, contributes to both peak accuracy and convergence speed. Trajectory analysis reveals an emergent implicit curriculum in which the scheduler concentrates budget on each domain during its active learning window.

References

[1] 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.

[2] Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

[3] Tianlei Chen, Jiao Ou, Ziyuan Liu, Ruiming Tang, Jian Liang, and Han Li. Counteraction-aware multi-teacher on-policy distillation for general capability recovery with domain preservation. *arXiv preprint arXiv:2605.27115*, 2026.

[4] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

[5] Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, 2026.

[6] Zhen Fang, Wenxuan Huang, Yu Zeng, Yiming Zhao, Shuang Chen, Kaituo Feng, Yunlong Lin, Lin Chen, Zehui Chen, Shaosheng Cao, and Feng Zhao. Flow-opd: On-policy distillation for flow matching models, 2026. URL https://arxiv.org/abs/2605.08063.

[7] 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.

[8] Wenjin Hou, Shangpin Peng, Weinong Wang, Zheng Ruan, Yue Zhang, Zhenglin Zhou, Mingqi Gao, Yifei Chen, Kaiqi Wang, Hongming Yang, et al. Uni-opd: Unifying on-policy distillation with a dual-perspective recipe. *arXiv preprint arXiv:2605.03677*, 2026.

[9] 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, 2025.

[10] Ijun Jang, Jewon Yeom, Juan Yeo, Hyunggyu Lim, and Taesup Kim. Stable on-policy distillation through adaptive target reformulation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42217–42227, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.2094. URL https://aclanthology.org/2026.findings-acl.2094/.

[11] Woogyeol Jin, Taywon Min, Yongjin Yang, Swanand Ravindra Kadhe, Yi Zhou, Dennis Wei, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. *arXiv preprint arXiv:2603.07079*, 2026.

[12] Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. Codeio: Condensing reasoning patterns via code input-output prediction. In *International Conference on Machine Learning*, pp. 34471–34489. PMLR, 2025.

[13] Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan ang 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.

[14] Yuying Li, Leqi Zheng, Yongzi Yu, Wenrui Zhou, Xuchang Zhong, Xing Hu, Jing Jin, Huangjie Yuan, and Tao Feng. Filter, then reweight: Rethinking optimization granularity in on-policy distillation. *arXiv preprint arXiv:2606.02684*, 2026.

[15] LLM-Core Xiaomi. Mimo-v2-flash technical report, 2026. URL https://arxiv.org/abs/2601.02780.

[16] Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, et al. Mopd: Multi-teacher on-policy distillation for capability integration in llm post-training. *arXiv preprint arXiv:2606.30406*, 2026.

[17] Mathematical Association of America. 2025 American Invitational Mathematics Examination I. https://artofproblemsolving.com/wiki/index.php/2025_AIME_I, 2025. Held on February 6, 2025.

[18] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[19] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

[20] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[21] Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

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

[23] Jiabin Shen, Guang Chen, and Chengjun Mao. Diagnosing and calibrating tool-call boundary drift in multi-teacher on-policy distillation, 2026. URL https://arxiv.org/abs/2607.07050.

[24] Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models, 2026. URL https://arxiv.org/abs/2604.00626.

[25] M3 Team, Chengfeng Dou, Fan Yang, Fei Li, Jiyuan Jia, Qiang Ju, Shuai Wang, Tianpeng Li, Xiangrong Zeng, Yijie Zhou, Hongda Zhang, Jinyang Tai, Linzhuang Sun, Peidong Guo, Yichuan Mo, Xiaochuan Wang, Hengfu Cui, and Zhishou Zhang. Baichuan-m3: Modeling clinical inquiry for reliable medical decision-making, 2026. URL https://arxiv.org/abs/2602.06570.

[26] Zhexu Wang, Yiping Liu, Yejie Wang, Wenyang He, Bofei Gao, Muxi Diao, Yanxu Chen, Kelin Fu, Flood Sung, Zhilin Yang, et al. Ojbench: A competition level code benchmark for large language models. *arXiv preprint arXiv:2506.16395*, 2025.

[27] Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation, 2026. URL https://arxiv.org/abs/2602.12125.

[28] Zhuolin Yang, Zihan Liu, Yang Chen, Wenliang Dai, Boxin Wang, Sheng-Chieh Lin, Chankyu Lee, Yangyi Chen, Dongfu Jiang, Jiafan He, et al. Nemotron-cascade 2: Post-training llms with cascade rl and multi-domain on-policy distillation. *arXiv preprint arXiv:2603.19220*, 2026.

[29] Qixiang Yin, Huanjin Yao, Yuchen Cai, Jianghao Chen, Ziyi Wang, Min Yang, Fei Su, and Zhicheng Zhao. H-opd: Confidence aware heterogeneous multi-teacher multimodal on-policy distillation, 2026. URL https://arxiv.org/abs/2607.02592.

[30] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2025.

[31] Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

[32] Dongxu Zhang, Zhichao Yang, Sepehr Janghorbani, Jun Han, Andrew Ressler II, Qian Qian, Gregory D Lyng, Sanjit Singh Batra, and Robert E. Tillman. Fast and effective on-policy distillation from reasoning prefixes. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25553–25569, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1276. URL https://aclanthology.org/2026.findings-acl.1276/.

[33] Zhiwei Zhang, Fei Zhao, Rui Wang, Zezhong Wang, Bin Liang, Jiakang Wang, Yao Hu, Shaosheng Cao, and Kam-Fai Wong. Robust tool use via fission-grpo: Learning to recover from execution errors. *arXiv preprint arXiv:2601.15625*, 2026.

[34] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[35] Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Contributors

Zechen Sun, Zhiwei Zhang, Fei Zhao, Juntao Li, Mu Chuan, Huayu Deng, Guojian Zhan, Wenliang Chen, Yao Hu, Min Zhang.

B Training Hyperparameters and Infrastructure

All main and ablation runs share the hyperparameters below; only the mixture-scheduling components (§3.2) differ across variants.

B.1 D^3-MOPD Scheduler

Table 3 lists the scheduler settings used for the D^3-MOPD composite-signal method.

Table 3: D^3-MOPD scheduler hyperparameters.

Symbol	Description	Value
n	Watcher update cadence (rollout steps)	10
W	Window length in the descent-velocity estimate (rollout steps)	10
R	Number of non-overlapping windows averaged in v_k	3
S_0	Seed length for the initial-KL normalizer $\mathrm{KL}_k^{(0)}$	5
–	EMA window applied to raw per-step KL	10
–	Numerical KL floor (ϵ_{KL}) in denominators	0.15
T	Softmax temperature in Eq. 6	0.5
ϵ	Per-domain mixture floor in Eq. 6	0.10
η	Batch-level jitter amplitude	0.30
–	File-poll interval of the watcher process (seconds)	300

B.2 Student Training

The student is trained with slime [35] on top of the Megatron backend; rollout inference is handled by an SGLang engine group co-located with the trainer. All runs use 7 nodes of GPUs: 4 trainer nodes, 2 SGLang rollout nodes, and 1 teacher node shared by K=4 teachers, connected via RDMA InfiniBand. Table 4 lists the training-loop hyperparameters, and Table 5 lists parallelism and performance settings.

Table 4: Training hyperparameters.

Setting	Value
Student initialization	Qwen3.6-35B-A3B
Number of teachers K	4
Total rollout steps	256
Rollout batch size B (prompts per step)	128
Responses sampled per prompt	4
Global mini-batch size	128
Gradient-accumulation micro-steps per rollout step	4
Optimizer	Adam
Learning rate	1×10^{-6}
Learning-rate schedule	constant
Weight decay	0.1
Adam (β_1, β_2)	$(0.9, 0.98)$
Policy-loss variant (advantage estimator)	CISPO [2]
Clip ratio (low, high)	$(0.2, 0.2)$
OPD reverse-KL coefficient	1.0
Max prompt length	16,384 tokens
Max response length	8,192 tokens
Rollout sampling temperature	1.0
Chat-template thinking mode	disabled

Table 5: Parallelism and performance settings (Megatron backend).

Setting	Value
Tensor-model parallel size	2
Context-parallel size	4
Expert-model parallel size	8
Max tokens per GPU per micro-batch	6,144
SGLang rollout: GPUs per engine	4

Throughput overhead. Because the watcher runs as a standalone process that reads the training log and writes updated ratios to a status file without requiring the trainer to pause, it adds no synchronization overhead. Over the full 256-step run the mean throughputs of D^3-MOPD (520 tokens/GPU/s) and vanilla MOPD (531 tokens/GPU/s) differ by only 2.1%, a gap attributable to the shift in sequence-length distribution as the scheduler redirects budget toward longer-response domains rather than to any computational overhead from the watcher.

C Best-Average Checkpoint Comparison

Table 2 reports the per-benchmark *peak* accuracy, where each benchmark's best value may come from a different checkpoint. Table 6 instead reports the accuracy at the single checkpoint with the highest average score, reflecting the performance of a single deployable model.

Table 6: Per-benchmark accuracy (%) and normalized score at the best-average checkpoint (vanilla MOPD: step 143; D^3-MOPD: step 95). The Teacher row reports each domain's dedicated expert model (one specialized model per domain, not a single model). Unlike Table 2, all values in each row come from a single checkpoint. Bold marks the column best, underline the runner-up.

	Math		Instruction Following		Code		Tool-use	
Method	AIME25	HMMT(N)	IFBench	IFEval	LCB-v6	OJB-C++	BFCL(B)	Norm.
Teacher	**76.2**	**72.3**	**52.1**	**91.9**	**64.8**	<u>28.5</u>	**67.0**	**1.00**
Student	69.8	70.7	30.0	86.8	60.0	25.9	57.5	0.00
Vanilla MOPD (step 143)	73.2	71.0	<u>45.7</u>	91.0	61.0	27.6	60.0	0.48
Δ(Vanilla − Teacher)	−3.0	−1.3	−6.4	−0.9	−3.8	−0.9	−7.0	−0.52
D^3-MOPD (step 95)	<u>74.0</u>	<u>71.4</u>	45.1	<u>91.2</u>	<u>62.5</u>	**29.7**	<u>62.5</u>	<u>0.73</u>
Δ(D^3-MOPD − Teacher)	−2.2	−0.9	−7.0	−0.7	−2.3	+1.2	−4.5	−0.27
Δ(D^3-MOPD − Vanilla)	+0.8	+0.4	−0.6	+0.2	+1.5	+2.1	+2.5	+0.25

Comparison with per-benchmark peaks. Compared with Table 2, the normalized scores drop for both methods (vanilla MOPD: 0.63 → 0.48; D^3-MOPD: 0.97 → 0.73), since no single checkpoint can maximize every benchmark at once. The drop is concentrated in benchmarks whose peaks occur far from the best-average step: for example, D^3-MOPD's HMMT(N) falls from 73.2 (per-benchmark peak) to 71.4 at step 95, indicating that HMMT(N) peaks at a later checkpoint. Nevertheless, D^3-MOPD at step 95 still outperforms vanilla MOPD at step 143 on six of seven benchmarks and achieves a notably higher normalized score (0.73 vs. 0.48), confirming that the improvement holds when deploying a single checkpoint.

D Generalization to 4B Student

To verify that D^3-MOPD generalizes across model scales, we replicate the training pipeline with Qwen3.5-4B [20] as the student. The four domain-specific teachers are obtained by GRPO-training the same model on the corresponding data used for the Qwen3.6-35B-A3B experiments. Training hyperparameters (Table 4) and scheduler settings (Table 3) follow the same configuration with minor scale-related adjustments.

Table 7: Per-benchmark accuracy (%) and normalized score with Qwen3.5-4B student (vanilla MOPD: step 159; D^3-MOPD: step 119). The Teacher row reports each domain's dedicated GRPO-trained 4B expert. Bold marks the column best, underline the runner-up.

	Math		Instruction Following		Code		Tool-use	
Method	AIME25	HMMT(N)	IFBench	IFEval	LCB-v6	OJB-C++	BFCL(B)	Norm.
Teacher	**63.4**	**66.4**	**55.9**	**85.3**	53.3	18.1	<u>63.0</u>	<u>1.00</u>
Student	47.8	51.7	35.9	85.3	39.4	13.8	51.0	0.00
Vanilla MOPD (step 159)	59.3	63.4	45.4	82.7	<u>53.7</u>	<u>18.5</u>	<u>63.0</u>	0.86
Δ(Vanilla − Teacher)	−4.1	−3.0	−10.5	−2.6	+0.4	+0.4	0.0	−0.14
D^3-MOPD (step 119)	<u>61.8</u>	<u>64.4</u>	<u>49.5</u>	<u>84.5</u>	**54.6**	**19.8**	**64.5**	**1.01**
Δ(D^3-MOPD − Teacher)	−1.6	−2.0	−6.4	−0.8	+1.3	+1.7	+1.5	+0.01
Δ(D^3-MOPD − Vanilla)	+2.5	+1.0	+4.1	+1.8	+0.9	+1.3	+1.5	+0.15

Results. Table 7 shows that the improvement pattern observed at the 35B-A3B scale carries over to the 4B student. D^3-MOPD outperforms vanilla MOPD on all seven benchmarks, with per-benchmark gains ranging from +0.9 (LCB-v6) to +4.1 (IFBench). The normalized score of D^3-MOPD (1.01) slightly surpasses the composite teacher ceiling (1.00), whereas vanilla MOPD reaches 0.86. Both methods surpass the domain-expert teachers on the code benchmarks (LCB-v6 and OJB-C++), and D^3-MOPD additionally exceeds the tool-use teacher on BFCL(B) by +1.5. The remaining gap to the teacher is largest on IFBench (−6.4), consistent with instruction following being the hardest domain for distillation at both model scales.

Convergence speed. D^3-MOPD reaches its best-average checkpoint 40 steps earlier than vanilla MOPD (step 119 vs. step 159), a relative reduction of 25% in the number of training steps needed to achieve peak average performance. This faster convergence aligns with the 35B-A3B result (step 95 vs. step 143, Table 6), indicating that dynamic domain scheduling accelerates convergence across model scales.

E Theoretical Analysis of Composite-Signal Allocation

We provide a simplified analysis showing that under an exponential decay model, the composite signal $s_k = \widetilde{\mathrm{KL}}_k \cdot v_k$ is proportional to the marginal reduction in the normalized gap, justifying its use as a greedy-optimal allocation criterion.

Exponential decay model. We model each domain's reverse-KL as decaying exponentially in its cumulative training exposure:

$$\mathrm{KL}_k(n_k) = \mathrm{KL}_k(0) \cdot e^{-\lambda_k n_k}, \tag{7}$$

where n_k is the cumulative sample count allocated to domain k and $\lambda_k > 0$ is a domain-specific learning rate. This captures the empirical observation (§2.2) that per-domain KL decreases roughly exponentially at domain-specific rates.

Proposition 1 *Under the exponential decay model (Eq. 7), with batch size B, window length W, uniform allocation $p_k = 1/K$, and small per-window changes ($\lambda_k B W / K \ll 1$), the composite signal satisfies*

$$s_k \;=\; \widetilde{\mathrm{KL}}_k \cdot v_k \;\approx\; \frac{B\,W}{K} \cdot \left| \frac{\partial \widetilde{\mathrm{KL}}_k}{\partial n_k} \right|. \tag{8}$$

Allocating samples to $\arg\max_k s_k$ *thus maximizes the instantaneous reduction in the average normalized gap* $\bar{g} = \frac{1}{K}\sum_k \widetilde{\mathrm{KL}}_k$.

Proof. The normalized gap is $\widetilde{\mathrm{KL}}_k = \mathrm{KL}_k(n_k)/\mathrm{KL}_k(0) = e^{-\lambda_k n_k}$, with marginal reduction

$$\left| \frac{\partial \widetilde{\mathrm{KL}}_k}{\partial n_k} \right| = \lambda_k\, e^{-\lambda_k n_k} = \lambda_k\, \widetilde{\mathrm{KL}}_k.$$

Under uniform allocation $p_k = 1/K$, each window of W rollout steps delivers BW/K samples to domain k. Substituting into Eq. 3, the single-window relative change is $\delta_k^{(i)} = e^{-\lambda_k BW/K} - 1$, constant across windows i. The descent velocity (Eq. 4) therefore reduces to $v_k = 1 - e^{-\lambda_k BW/K}$. A first-order expansion yields $v_k \approx \lambda_k BW/K$, giving

$$s_k = \widetilde{\mathrm{KL}}_k \cdot v_k \;\approx\; \frac{BW}{K} \cdot \lambda_k \widetilde{\mathrm{KL}}_k \;=\; \frac{BW}{K} \cdot \left| \frac{\partial \widetilde{\mathrm{KL}}_k}{\partial n_k} \right|.$$

Since BW/K is constant across domains, $\arg\max_k s_k = \arg\max_k |\partial \widetilde{\mathrm{KL}}_k / \partial n_k|$. □

Interpretation. Proposition 1 shows that the composite signal ranks domains by the magnitude of their normalized marginal KL reduction. The initial-KL normalization in $\widetilde{\mathrm{KL}}_k$ (Eq. 2) is essential: without it, domains with large absolute KL (e.g., IF, whose KL is $\sim$65$\times$ that of Math) would dominate the allocation regardless of their relative improvement rate. The first-order condition $\lambda_k BW/K \ll 1$ requires per-window KL changes to be small, which is satisfied in practice by the short window length ($W = 10$ steps).

Practical robustness. The softmax-floor mapping (Eq. 6) smooths the greedy $\arg\max_k s_k$ allocation: the temperature T distributes samples across domains in proportion to their exponential signal strength rather than concentrating on a single domain, while the floor ϵ guarantees every domain retains a minimum sample rate, preventing catastrophic forgetting of domains that have temporarily plateaued but may resume descent later in training.

Remark on the exponential decay assumption. The exponential decay model is a simplifying assumption that we adopt to derive the proportionality between s_k and the marginal KL reduction in closed form. The method itself does not require this assumption to hold exactly, because the composite signal is self-correcting by construction: if a domain deviates from exponential decay and enters an unexpected plateau, its descent velocity v_k drops toward zero and automatically reduces its allocation regardless of the underlying functional form. The theoretical analysis therefore serves as a design rationale for choosing the gap-velocity product as the allocation criterion rather than a necessary condition for the method to produce effective schedules.

F Composite Signal Trajectories

Figure 7 visualizes how the composite signal $\tilde{s}_k$ and the normalized KL $\widetilde{\mathrm{KL}}_k$ evolve for each domain over the full 256-step training run. Panel (e) shows the corresponding allocation ratio p_k produced by the scheduler.

The scheduler concentrates budget on whichever domain currently exhibits both a large remaining gap and rapid descent, producing a sequence of distinct allocation phases. In the first phase (steps 20–55), IF receives the highest signal because its initial KL gap is the largest among all domains and it descends rapidly once training begins, causing the allocation to reach approximately 38% while the other three domains receive near-floor shares. As IF's descent velocity slows around step 55, Math takes over as the dominant domain and holds $\tilde{s}_{\text{math}} = 1.0$ for roughly 60 consecutive steps, during which its normalized KL drops steadily from 0.42 to 0.25. The scheduler then redirects budget to Tool-use (steps 120–160) and finally returns to IF (steps 180–250) as IF's accumulated gap reopens after an extended period of reduced allocation.

The self-correcting property of the composite signal is visible in the interplay between the shaded signal regions and the KL curves within each panel. When a domain's signal drops to zero and it receives only the floor allocation $\epsilon = 0.10$, its normalized KL tends to plateau or rise (for example, IF's normalized KL climbs from 0.29 to 0.65 during steps 65–120), which eventually increases the remaining gap component and restores a nonzero signal in a later update. This feedback loop ensures that no domain is permanently neglected even without explicit revisitation logic in the scheduler.

G Ablation Domain Ratio Trajectories

Figure 8 shows the per-domain sampling ratio of each ablation variant alongside vanilla MOPD and D^3-MOPD over 256 rollout steps.

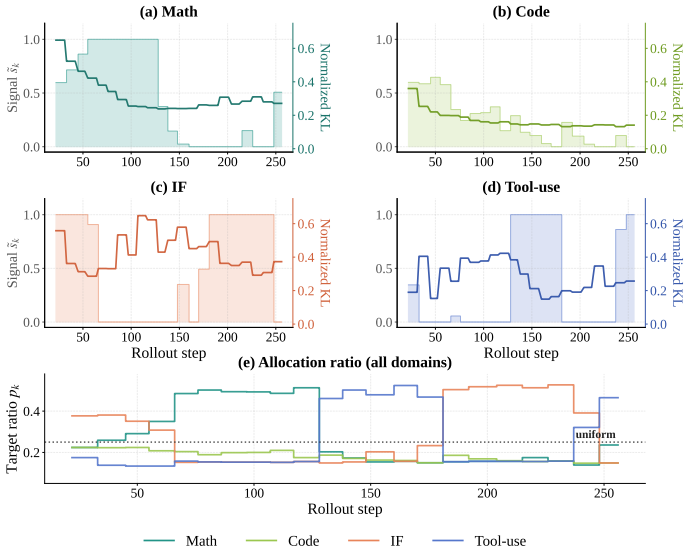

Figure 7: Per-domain composite signal and normalized KL over 256 rollout steps, with the resulting allocation ratio in panel (e). The scheduler produces a sequence of phase transitions that track the domain with the largest remaining gap and fastest descent at each point in training.

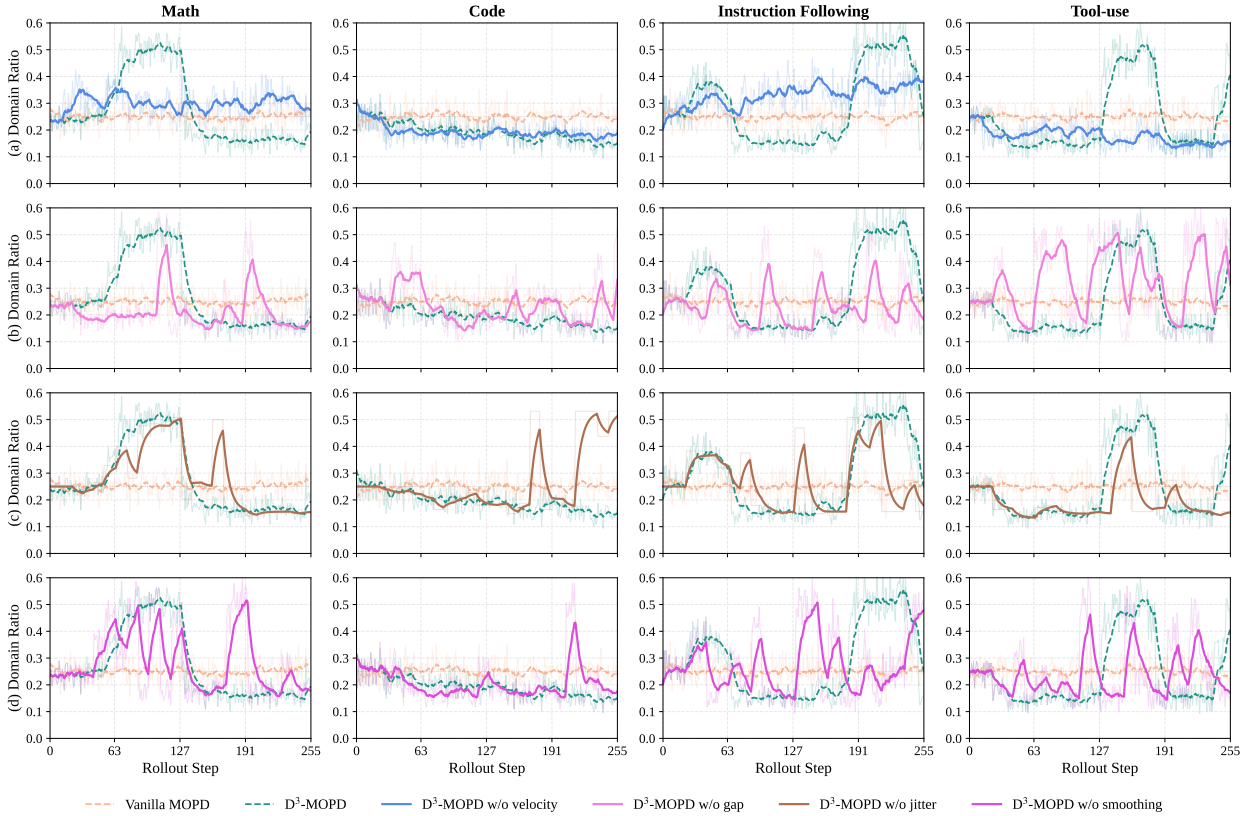

Figure 8: Per-domain sampling ratio of each ablation variant vs. D^3-MOPD and vanilla MOPD over 256 rollout steps. (a) w/o velocity, (b) w/o gap, (c) w/o jitter, (d) w/o smoothing.

- **w/o velocity (row a).** Without descent velocity, the scheduler relies solely on the KL gap. The ratios are flatter than D^3-MOPD: Math rises to $\sim$0.30 but never reaches the $\sim$0.50 peak of D^3-MOPD, and the late-

stage shift toward IF and Tool-use is weaker. Because the gap signal alone cannot distinguish an actively descending domain from one plateaued at a high KL, the scheduler spreads budget uniformly.

- **w/o gap (row b).** Without the remaining gap, the scheduler uses only descent velocity. Early in training all domains descend rapidly, so the velocity signal cannot differentiate them, producing large oscillations before step 63. Later, IF is identified as the fastest-descending domain and heavily upweighted, but the lack of gap information causes abrupt ratio swings (e.g., Math drops to $\sim$0.10 around step 100 before rebounding). This instability explains why the w/o gap variant peaks latest at step 255 (Table 1).
- **w/o jitter (row c).** The overall trajectory shape is similar to D^3-MOPD, confirming that jitter does not alter the scheduler's long-run allocation. However, the ratio curves show sharper step-like transitions between watcher updates, most visible in Code near step 200 and Tool-use near step 240. The corresponding accuracy drops on code benchmarks (Table 1) suggest that this reduced per-batch variation hurts domains near their r-KL plateau.
- **w/o smoothing (row d).** A single observation window (R=1) makes the velocity estimate more reactive but noisier. The ratio curves oscillate visibly more than D^3-MOPD (R=3), with frequent spikes in Math and Code around steps 60–130. The overall allocation pattern nonetheless remains close to D^3-MOPD, which explains why this variant still achieves 62.17 (+0.81 in Table 1).