PAC: Progress-Augmented Advantage Curriculum for Multi-Task Reinforcement Learning of LLMs

Yuanqiang Yu, Yanzhao Zheng, Zhentao Zhang, Tianze Xu, Chao Ma, Jihuai Zhu Jiashun Liu, Xinle Deng, Baohua Dong*, Hangcheng Zhu, Ruohui Huang
Alibaba Group, Hangzhou, China
{yuyuanqiang.yyq,zhengyanzhao.zyz}@alibaba-inc.com
{zhangzhentao.zzt,xutianze.xtz,mc524716}@alibaba-inc.com
{zhujihuai.zjh,baohua.dbh}@alibaba-inc.com
{linran.lr09,wentong}@alibaba-inc.com

Abstract

Reinforcement learning (RL) is used to improve the reasoning abilities of LLMs, while training data span heterogeneous tasks. However, most RL post-training pipelines rely on fixed or manually designed task mixtures, even though task usefulness changes as training progresses. Online curriculum methods often define learnability by update magnitude, ignoring whether the update translates into reward gains, which can misallocate rollout budget toward tasks with large but ineffective updates. We propose **PAC**, a Progress-Augmented Advantage Curriculum for multi-task RL of LLMs that combines two task-level signals: advantage-derived learnability, which measures the magnitude of the policy update a task can induce, and recent reward gains, which show whether those updates have improved task performance. A Bayesian Thompson Sampling controller uses these signals to allocate rollouts across tasks during GRPO training. We evaluate **PAC** under two settings: a multi-level reasoning setting and a multi-domain reasoning setting. **PAC** improves sample efficiency and final performance: it reaches comparable validation scores with fewer rollout steps and achieves higher final averages than random sampling and advantage-based curriculum baselines in both settings. These results show that jointly tracking advantage signals and actual reward gains yields an effective online curriculum for LLM post-training.

1 Introduction

In practical LLM post-training, reinforcement learning (RL) is increasingly applied to heterogeneous task mixtures rather than to a single fixed task. Recent LLM training reports, including DeepSeek-R1 (Guo et al., 2025), Qwen3 (Yang et al., 2025a), and Tulu 3 (Lambert et al., 2025), describe RL post-training over mixtures of reasoning, coding, STEM, instruction following, multilingual, or multimodal data. However, these reports do not focus on adapting the task mixture online as the policy changes. This gap raises a central curriculum question: how should a trainer allocate limited rollout budget across tasks during multi-task RL post-training?

This question is especially consequential for Group Relative Policy Optimization (GRPO), a widely used RL post-training algorithm with verifiable rewards. Each update samples multiple responses to the same prompt, scores them, and derives relative advantages from the resulting rewards (Shao et al., 2024), making rollouts a major training cost. Recent GRPO variants improve the optimizer through revised clipping, sampling, loss design, bias correction, or sequence-level policy ratios (Yu et al., 2025; Liu et al., 2025; Zheng et al., 2025). These optimizer-level advances are complementary to our focus: they generally assume a fixed task distribution and do not adapt the task mixture itself.

The core difficulty is that task utility is non-stationary during training. Easy tasks may stop providing useful learning signal once performance saturates; hard tasks may provide little actionable feedback early in training; and ability-dependent tasks may become useful only after their prerequisite abilities are learned. Uniform sampling ignores these dynamics, while hand-designed or difficulty-based curricula require difficulty labels, filtering rules, or fixed schedules that often transfer poorly across settings (Bengio et al., 2009; Narvekar et al., 2020; Parashar et al., 2026; Shi et al., 2026; Song et al., 2025).

Recent online data-selection and curriculum methods adapt using learnability, difficulty, posterior uncertainty, gradient-based signals, or influence estimates (Foster et al., 2025; Bae et al., 2026; Qu et al., 2026; Shen et al., 2026; Zeng et al., 2026; Zhu et al., 2026). While these signals enable online adaptation, they do not directly measure the

*Corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 25825–25843
October 24-29, 2026 ©2026 Association for Computational Linguistics

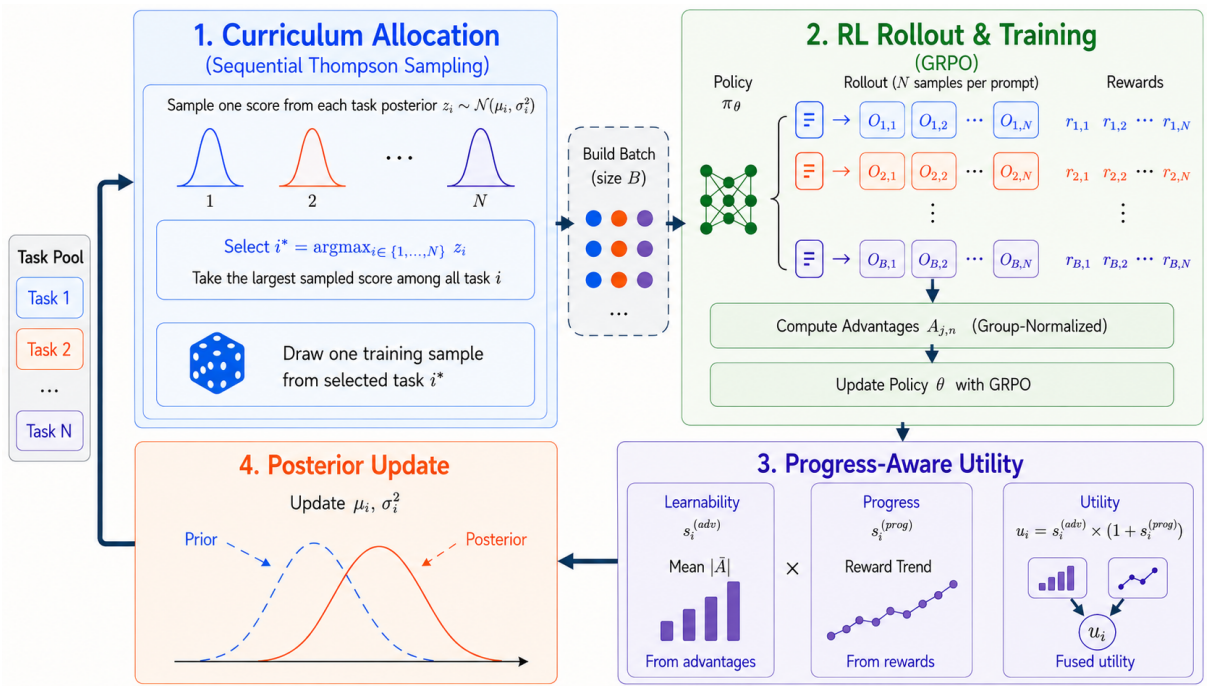


Figure 1: **Illustration of PAC for multi-task RL post-training.** At each iteration, a Thompson Sampling controller allocates rollout budget across task arms to construct a mixed batch. The sampled responses are evaluated with verifiable reward functions, from which group-normalized advantages are estimated for the GRPO update. **PAC** fuses advantage-derived learnability with reward-derived progress into utility observations, updates task posteriors, and uses them to allocate the next rollout batch.

allocation target: whether additional rollouts on a task translate into reward improvement. A task can be uncertain, difficult, or update-inducing without producing reward gains under the current policy. In LLM post-training, recent bandit curricula often use absolute advantage as the bandit reward (Chen et al., 2025; Wang et al., 2025). However, the absolute advantage primarily reflects the magnitude of the policy update a task can induce; it does not indicate whether the update translates into higher reward. In heterogeneous task mixtures, this distinction matters: high-variance tasks can be overprioritized even when their expected rewards are not improving, while tasks with moderate advantages but steadily improving rewards may be undersampled.

To address this limitation, we propose **PAC**, a Progress-Augmented Advantage Curriculum for multi-task RL post-training of LLMs. As shown in Figure 1, **PAC** estimates task utility from two complementary online signals: advantage-derived learnability proxies the update potential available from a task, while reward-derived progress measures whether recent updates translate into reward gains. **PAC** fuses the two signals into utility observations and uses a Bayesian Thompson Sampling controller to maintain task-level beliefs and allocate future rollouts under uncertainty. Our contributions are threefold. First, we highlight a limitation of advantage-only task utility for online curricula in heterogeneous LLM RL mixtures. Second, we introduce **PAC**, a progress-augmented Thompson Sampling curriculum for adaptive rollout allocation. Third, we validate **PAC** in multi-level and multi-domain reasoning settings, where it improves sample efficiency and final performance over curriculum baselines.

2 Related Work

Curriculum Learning for SFT. Curriculum learning studies how ordering or weighting examples and tasks affects optimization and generalization (Bengio et al., 2009; Narvekar et al., 2020). In supervised fine-tuning (SFT) of LLMs, curricula usually rely on static mixtures or precomputed properties such as task balance, data quality, difficulty, and instruction diversity (Sanh et al., 2022; Wei et al., 2022; Wang et al., 2022, 2023; Longpre et al., 2023; Zelikman et al., 2022; Xu et al., 2024; Luo et al., 2025). Because targets are fixed and task utility is largely determined before training, these methods do not address the online allocation

problem in RL post-training, where task usefulness shifts as the policy evolves.

Curriculum Learning for RL. RL curricula adapt tasks or environments using episodic return, learning progress, or estimated future learning potential (Portelas et al., 2020; Graves et al., 2017; Matiisen et al., 2020). Prioritized Level Replay revisits levels with high learning potential (Jiang et al., 2021); T3S combines task-specific feature selection with a scheduler, whereas Distral focuses on knowledge transfer through a shared distilled policy (Yu et al., 2023; Teh et al., 2017). Most closely, Hard Tasks First introduces Scheduled Multi-Task Training (SMT), which prioritizes tasks according to difficulty estimates derived from returns and policy entropy and periodically resets selected network parameters (Cho et al., 2024). These RL curriculum methods are complementary to ours but do not directly target online allocation over LLM task mixtures.

Curriculum Learning for LLM Post-Training. Recent RLHF and reasoning-oriented systems train on heterogeneous mixtures with PPO or GRPO, but typically rely on fixed or manually designed task mixtures rather than an explicit online allocation rule (OpenAI, 2024; Guo et al., 2025; Lambert et al., 2025). Existing post-training curricula either select frontier or intermediate-difficulty prompts, replay online data, or train samplers from advantage-based bandit rewards and Bayesian online task selection (Foster et al., 2025; Bae et al., 2026; Sun et al., 2025; Muhtar et al., 2026; Chen et al., 2025; Shen et al., 2026). **PAC** is closest to advantage-based online allocation, where advantage magnitude provides a proxy for update strength but ignores whether the update yields reward improvement. To address this gap, **PAC** combines advantage-derived learnability with reward-derived progress in a Thompson Sampling controller for rollout allocation.

3 PAC: Progress-Augmented Advantage Curriculum

We describe **PAC** as an online controller for multi-task GRPO post-training. The method treats task selection as a non-stationary allocation problem, where reward and advantage signals from recent rollouts update beliefs about which tasks are useful for the current policy. We first define the multi-task training setting and the allocation objective. The following sections then introduce the task utility, the Bayesian curriculum controller, and the full training algorithm.

3.1 Problem Formulation

We consider RL post-training of an LLM policy π_θ over N task datasets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$. At training step t, the curriculum controller determines the allocation of training samples over the task datasets. We represent this allocation by a task-sampling distribution $\boldsymbol{p}^{(t)} = (p_1^{(t)}, \ldots, p_N^{(t)})$ over the N tasks, with $p_i^{(t)} \geq 0$ and $\sum_{i=1}^{N} p_i^{(t)} = 1$. Each component $p_i^{(t)}$ denotes the probability that one training sample is assigned to task i; in **PAC**, this allocation is induced implicitly by sequential Thompson Sampling.

For a selected prompt x, the current policy generates G rollouts $\{o_j\}_{j=1}^{G} \sim \pi_\theta(\cdot \mid x)$, and each rollout o_j is scored by a verifiable reward r_j. Under an allocation $\boldsymbol{p}^{(t)}$, a batch of size B is formed by repeatedly sampling a task $i \sim \boldsymbol{p}^{(t)}$ and then sampling a prompt $x \sim \mathcal{D}_i$ from the selected task. Thus, a single batch may contain training samples from several tasks. For each rollout o_j, we compute the group-normalized advantage as

$$\hat{A}_j = \frac{r_j - \mathrm{mean}(\{r_1, \ldots, r_G\})}{\mathrm{std}(\{r_1, \ldots, r_G\}) + \epsilon},$$

where the mean and standard deviation are computed over the G rewards sampled for the same prompt, and $\epsilon > 0$ is a small stability constant. Following standard policy-gradient methods, the update term for a prompt x is

$$g_x(\theta) = \mathbb{E}_{o_{1:G} \sim \pi_\theta(\cdot|x)} \left[\frac{1}{G} \sum_{j=1}^{G} \hat{A}_j \, \nabla_\theta \log \pi_\theta(o_j \mid x) \right]. \tag{1}$$

Averaging $g_x(\theta)$ over prompts in task i yields the task-conditioned gradient $g_i(\theta) = \mathbb{E}_{x \sim \mathcal{D}_i}[g_x(\theta)]$, and the mixed-batch GRPO gradient is

$$\nabla_\theta \mathcal{J}_t(\theta) \;=\; \sum_{i=1}^{N} p_i^{(t)} g_i(\theta). \tag{2}$$

The goal is to update θ while adapting the task allocation, thereby spending the limited rollout budget on tasks that are most useful for the current policy.

We model this allocation problem as a non-stationary multi-armed bandit. Each task is an arm, and pulling arm i assigns one training sample to $\mathcal{D}_i$. Arm utilities change as the policy is updated: a

task may become learnable, saturate, or remain too difficult for the current policy. **PAC** uses this feedback loop to adapt the curriculum. GRPO rollouts produce rewards and advantages; these signals are summarized as task-level utilities, and the Bayesian controller updates its posterior before forming the next rollout batch.

3.2 Online Task Utility Estimation

The ideal curriculum would allocate the next training sample to the task that yields the largest short-horizon improvement. This marginal gain cannot be measured directly online, because a batch reveals feedback only for the tasks it contains and does not expose cross-task transfer. **PAC** therefore approximates the latent marginal value of each arm using observable proxies; transfer effects are reflected later in the reward histories of affected tasks.

Let $\mathcal{V}_i(\theta)$ be the expected reward of task i under π_θ, and let $\Delta\theta_i^{(t)}$ be the marginal GRPO update induced by assigning one training sample to task i at step t. We define arm utility as the expected short-horizon gain in the task-specific reward objective. A first-order Taylor expansion around θ_t gives

$$\begin{aligned} u_i^{\star(t)} &= \mathbb{E}\Big[\mathcal{V}_i(\theta_t + \Delta\theta_i^{(t)}) - \mathcal{V}_i(\theta_t) \,\Big|\, x \sim \mathcal{D}_i\Big] \\ &\approx \mathbb{E}\Big[\nabla_\theta \mathcal{V}_i(\theta_t)^\top \Delta\theta_i^{(t)} \,\Big|\, x \sim \mathcal{D}_i\Big]. \end{aligned} \tag{3}$$

Decomposing $\Delta\theta_i^{(t)} = \|\Delta\theta_i^{(t)}\|_2 \hat{\Delta}\theta_i^{(t)}$ into magnitude and direction separates the inner product into two interpretable factors. The directional term $\nabla_\theta \mathcal{V}_i(\theta_t)^\top \hat{\Delta}\theta_i^{(t)}$ measures reward improvement per unit update and motivates the conversion factor $\textbf{Convert}_i$. The magnitude term $\|\Delta\theta_i^{(t)}\|_2$ captures the size of the task-induced update and motivates the learnability factor $\textbf{Learnability}_i$. Accordingly,

$$u_i^{(t)} \propto \underbrace{\mathbb{E}\Big[\nabla_\theta \mathcal{V}_i(\theta_t)^\top \hat{\Delta}\theta_i^{(t)}\Big]}_{\textbf{Convert}_i} \cdot \underbrace{\mathbb{E}\Big[\|\Delta\theta_i^{(t)}\|_2\Big]}_{\textbf{Learnability}_i}. \tag{4}$$

This factorization motivates a utility that combines update magnitude with evidence that those updates translate into reward gains. Because neither factor is directly observable online, we approximate each from signals already produced during training: $\textbf{Learnability}_i$ from advantage magnitudes (Section 3.2.1), and $\textbf{Convert}_i$ from recent reward gains over a local window (Section 3.2.2).

3.2.1 Advantage-Derived Learnability

The GRPO policy-gradient loss is a sum of policy-gradient terms weighted by group-normalized advantages. Appendix A gives the supporting bound: when the log-probability gradient norms $\|\nabla_\theta \log \pi_\theta(o_j \mid x)\|$ vary smoothly across tasks, differences in update magnitude are largely explained by differences in advantage magnitude. Prompts whose sampled responses receive clearly separated rewards provide stronger learning signals than prompts whose responses receive nearly identical rewards. At training step t, let $\mathcal{H}_t = \{t-W, \ldots, t-1\}$ denote the history window, and let $\mathcal{X}_i^{(t')}$ be the prompts from task i at historical step $t' \in \mathcal{H}_t$. For each task i, we estimate learnability by first computing a step-level mean absolute advantage and then averaging it over this window:

$$\begin{aligned} a_i^{(t')} &= \frac{1}{|\mathcal{X}_i^{(t')}|} \sum_{x \in \mathcal{X}_i^{(t')}} \frac{1}{G} \sum_{j=1}^{G} \left|\hat{A}_{t',x,j}\right|, \\ s_i^{(\mathrm{adv})} &= \frac{1}{W} \sum_{t' \in \mathcal{H}_t} a_i^{(t')}. \end{aligned} \tag{5}$$

where $\hat{A}_{t',x,j}$ is the group-normalized advantage of rollout j for prompt x at step t'. Following Eq. (4), we use $s_i^{(\mathrm{adv})}$ to estimate $\textbf{Learnability}_i$, which reflects the advantage-driven update that task i can still provide.

3.2.2 Reward-Derived Progress

Advantage magnitude alone does not show whether the policy is improving on a task: a task can induce large GRPO updates even when its task-level expected reward shows no measurable upward trend. We therefore pair advantage-derived learnability with a signal that tracks recent reward gains. At the current controller step t, this improvement cannot be estimated reliably from a single mixed batch, since task-level reward averages are noisy and some tasks may receive only a few samples. **PAC** therefore looks back over the previous W training steps and estimates whether each task's mean reward has been increasing.

Using the same history window $\mathcal{H}_t$, let $\bar{r}_i^{(t')}$ be the step-level mean reward of task i at historical step $t' \in \mathcal{H}_t$, where the mean is taken over the per-rollout rewards r_j defined in Section 3.1. To estimate recent progress, we fit a linear regression model via least squares of the form

$$\bar{r}_i(\tau_{t'}) = b_i + k_i \tau_{t'}, \tag{6}$$

where $\tau_{t'}$ is the position of historical step t' normalized to $[0, 1]$ within the window. Let $\hat{k}_i$ be the fitted slope. We normalize the fitted slopes across tasks:

$$s_i^{(\text{prog})} = \frac{\hat{k}_i}{\sum_{j=1}^{N} |\hat{k}_j| + \epsilon}, \tag{7}$$

where ϵ is a small stability constant. Under the factorization in Eq. (4), $s_i^{(\text{prog})}$ estimates the relative reward trend component: a task receives a larger trend adjustment when its reward improves faster than rewards on the other tasks in the current training phase.

3.2.3 Progress-Modulated Utility

To combine the two signals, **PAC** takes advantage-derived learnability as the base utility and rescales it by normalized progress. Setting $\textbf{Learnability}_i = s_i^{(\text{adv})}$ and $\textbf{Convert}_i = 1 + s_i^{(\text{prog})}$ gives

$$u_i^{(t)} = \left(1 + s_i^{(\text{prog})}\right) \cdot s_i^{(\text{adv})}. \tag{8}$$

Here, $s_i^{(\text{adv})}$ estimates the magnitude of the task-induced GRPO update, while $1 + s_i^{(\text{prog})}$ modulates this magnitude using recent reward gains. The unit offset makes the modifier neutral when the normalized progress estimate is near zero, preserving informative advantage signals.

3.3 Bayesian Curriculum Controller

The fused utility $u_i^{(t)}$ in Eq. (8) is noisy and is observed with unequal reliability across arms, since different arms receive different numbers of samples in each batch. **PAC** therefore maintains a Gaussian belief over the latent utility v_i of each task arm:

$$v_i \sim \mathcal{N}(\mu_i, \sigma_i^2), \tag{9}$$

where the posterior mean μ_i summarizes the current utility estimate and the posterior variance σ_i^2 quantifies the remaining uncertainty that drives exploration in Thompson Sampling. Let $c_i^{(t)}$ be the number of training samples drawn from task i at step t, $\bar{c} = B/N$ the count under uniform allocation, and $\rho_i^{(t)} = c_i^{(t)}/\bar{c}$ the relative amount of evidence for task i in that batch. We treat $\rho_i^{(t)}$ as the observation precision of $u_i^{(t)}$, so that utility estimates supported by more samples are weighted more heavily, while sparsely sampled arms retain higher posterior uncertainty. Under this Gaussian–Gaussian observation model, the conjugate update is a precision-weighted average:

$$\begin{aligned} \eta_i &= \frac{1}{\sigma_i^2} + \rho_i^{(t)}, \\ \mu_i^+ &= \frac{\frac{\mu_i}{\sigma_i^2} + \rho_i^{(t)} u_i^{(t)}}{\eta_i}, \\ (\sigma_i^2)^+ &= \eta_i^{-1}. \end{aligned} \tag{10}$$

The update therefore gives more influence to observations supported by more samples, while preserving higher posterior uncertainty for arms that have been sampled less often. Task utilities are also non-stationary: an arm that is currently informative may saturate after a few updates, and a previously stalled arm may become learnable as the policy improves. To prevent the posterior from collapsing onto stale evidence, we apply non-stationary variance inflation after each update:

$$\mu_i \leftarrow \mu_i^+, \qquad \sigma_i^2 \leftarrow (\sigma_i^2)^+ + \delta^2, \tag{11}$$

where the inflation variance δ^2 keeps the posterior variance bounded away from zero, which prevents any single arm from being overprioritized early in training and preserves responsiveness to later shifts in arm utility.

With the posterior in place, the controller builds the next batch of size B by sequential Thompson Sampling. For each training sample in the batch, the controller draws one utility value $\tilde{v}_j \sim \mathcal{N}(\mu_j, \sigma_j^2)$ for every arm j, selects $i = \arg\max_j \tilde{v}_j$, and samples one prompt from $\mathcal{D}_i$; repeating this draw B times produces the full batch. This per-sample draw lets a single batch mix prompts from multiple arms in proportion to their posteriors, instead of locking the batch onto the arm with the largest current estimate. This design exploits arms with high posterior mean while keeping exploration proportional to posterior variance; the variance inflation in Eq. (11) helps preserve opportunities to revisit arms whose utility shifts later in training.

3.4 Training Procedure

Algorithm 1 summarizes the training loop, which proceeds in two stages. At the start of training, reward and advantage statistics for each arm are based on only a few rollouts, so adapting the sampling distribution immediately would amplify the noise in those early estimates. **PAC** therefore opens

Algorithm 1 **PAC: Progress-Augmented Advantage Curriculum**

Require: Training set with N tasks $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$; LLM policy π_θ with parameters θ; Batch size B; Total training steps T; Cold-start length T_{cold}

Initialize $(\mu_i, \sigma_i^2) \leftarrow (0, 1) \quad \forall i \in \{1, 2, \ldots, N\}$
for $t \leftarrow 0$ **to** $T - 1$ **do**
  $\mathcal{B}_t \leftarrow \emptyset$
  while $|\mathcal{B}_t| < B$ **do**
    if $t < T_{\text{cold}}$ **then**
      Select task i uniformly from $\{1, \ldots, N\}$
    else
      Draw $\tilde{v}_j \sim \mathcal{N}(\mu_j, \sigma_j^2) \quad \forall j \in \{1, \ldots, N\}$
      Select task $i \leftarrow \arg\max_j \tilde{v}_j$
    end if
    Sample prompt x uniformly from $\mathcal{D}_i$
    $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{x\}$
  end while
  Run π_θ on each $x \in \mathcal{B}_t$ to generate rollouts $\mathcal{T}$ and compute rewards $\mathbf{r}$
  Estimate advantages $\hat{A}$ and update π_θ with GRPO
  for all task $i \in \{1, 2, \ldots, N\}$ **do**
    Compute fused utility $u_i^{(t)}$ using Eq. (8)
    Update (μ_i, σ_i^2) with $u_i^{(t)}$ using Eq. (10) and Eq. (11)
  end for
end for
return Fine-tuned LLM π_θ

with a cold-start stage of T_{cold} steps during which batches are drawn uniformly over arms. The controller still computes the fused utility $u_i^{(t)}$ and updates the Gaussian posterior throughout this stage, so every arm accumulates comparable initial evidence before Thompson Sampling becomes active.

After the cold-start stage, each training step alternates between a policy update and a controller update. The controller first builds a batch of size B by sequential Thompson Sampling (Section 3.3), and the policy then generates GRPO rollouts on the selected prompts and receives verifiable rewards. The resulting reward and advantage statistics are appended to the history window, after which **PAC** recomputes the fused utility via Eq. (8), applies the precision-weighted posterior update in Eq. (10), and applies the variance inflation in Eq. (11) before the next step.

Across the two stages, **PAC** progressively reallocates rollout budget toward tasks that show both an advantage-derived learning signal and measurable reward improvement. Thompson Sampling trades off exploitation of high-utility tasks against exploration under posterior uncertainty, while the variance inflation in Eq. (11) is intended to maintain responsiveness as task utilities evolve during training.

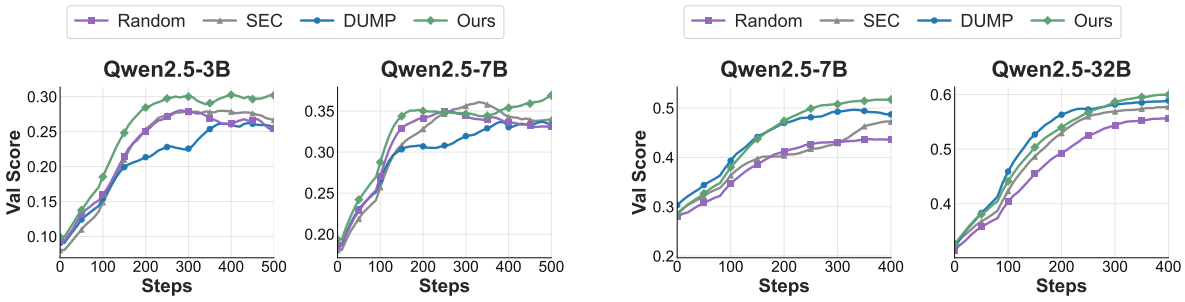
4 Experiments

4.1 Experimental Setup

Datasets and Evaluation. We evaluate **PAC** in two multi-task RL settings: a multi-level reasoning setting based on Chen et al. (2025) and a multi-domain mixture spanning mathematics, code generation, and symbolic logic puzzles.

Multi-level reasoning setting. Following the multi-level task setup of SEC (Chen et al., 2025), this setting includes Countdown, Zebra, and ARC (Abstraction and Reasoning Corpus) (Chollet, 2019). Countdown requires the model to combine a given set of integers and arithmetic operators to reach a target number; difficulty is controlled by the number of input integers. Zebra is a constraint-satisfaction logic puzzle in which entities must be assigned properties from textual clues; difficulty increases with the number of entities and properties. ARC provides input-output string transformation examples and asks the model to infer the underlying rule for an unseen case; input-string length controls difficulty. For each task, we use the training portions of the simple, medium, and hard splits for RL post-training. We evaluate on disjoint held-out validation portions at all four difficulty levels; the main results use the extremely hard level to measure transfer to harder problems, while Appendix B.4 reports the complete per-difficulty breakdown. We run this setting with Qwen2.5-3B and Qwen2.5-7B (Yang et al., 2025b).

Multi-domain reasoning setting. For the math domain, we train on DAPO-Math-17k, a mathematical reasoning corpus designed for rule-based answer verification in RL training (Yu et al., 2025). For the code domain, we train on Code-R1-12k, where each example provides a programming prompt with unit-test-based ground truth for automatic validation; following OMNI-THINKER (Li et al., 2025), we filter out prompts longer than 1,024 tokens. For the logic domain, we use the 3-, 4-, and 5-character splits of the K&K puzzle dataset, where each character is either a knight who always tells the truth or a knave who always lies, and the goal is to infer the identity of each character (Wang et al., 2025). For validation, math is evaluated on MATH500 (Hendrycks et al., 2021; Lightman et al., 2023), AMC22–23, and AIME24; code is evaluated by pass@1 on BigCodeBench (Zhuo et al., 2025); and K&K is evaluated on the held-out 6-character split. Because this setting requires stronger base capabilities across mathematics, code



(a) **Multi-level reasoning setting.**

(b) **Multi-domain reasoning setting.**

Figure 2: **Validation score over training progress for the two multi-task RL settings.** Figure (a) shows the multi-level reasoning setting with Qwen2.5-3B and Qwen2.5-7B, evaluated on the extremely hard validation splits of Countdown, Zebra, and ARC. Figure (b) shows the multi-domain reasoning setting with Qwen2.5-7B and Qwen2.5-32B, evaluated on MATH500, AMC22–23, AIME24, BigCodeBench, and the held-out 6-character K&K split. In both figures, "Val Score" denotes the mean of the task-level validation scores.

generation, and symbolic logic, we use larger backbones: Qwen2.5-7B and Qwen2.5-32B. Representative prompt examples for all datasets used in the two settings are provided in Appendix E.

Baselines. We compare against one single-task reference and three curriculum baselines. **ST** (Single Task) trains each task or domain separately and evaluates only on its matching validation set. **Random** samples training examples uniformly from the available tasks or task splits. **SEC** implements the Self-Evolving Curriculum of Chen et al. (2025), which formulates curriculum selection as a non-stationary multi-armed bandit over task categories, uses average absolute advantage as the arm reward, and selects categories with Boltzmann Sampling. **DUMP** follows the same advantage-driven principle as SEC but performs distribution-level scheduling with UCB-based sampling (Wang et al., 2025). Additional implementation details are provided in Appendix B.

4.2 Results and Discussion

Figure 2 reports the validation score over training progress, and Tables 1–2 report final validation scores. All numbers in tables and all curves in figures are averaged over three runs with different random seeds. ST is a single-task reference and is excluded from highlighting.

PAC improves both sample efficiency and final performance. In Figure 2, **PAC** reaches the final average scores of the strongest curriculum baselines with fewer rollout steps. In the multi-level setting, **PAC** matches the final average of SEC after about 170 steps with Qwen2.5-3B and 143 steps with Qwen2.5-7B, whereas SEC reaches its own final average at about 225 and 227 steps, corresponding to sample-efficiency gains of roughly $1.3\times$ and $1.6\times$. In the multi-domain setting, **PAC** reaches the final average of DUMP earlier for both Qwen2.5-7B and Qwen2.5-32B, corresponding to approximately $1.2\times$ and $1.3\times$ sample-efficiency gains, respectively, while maintaining higher final averages. This behavior is consistent with the utility design: advantage magnitude identifies tasks that can still provide informative updates, while reward-derived progress discounts updates that no longer lead to reward gains.

Multi-level results show more balanced transfer to harder splits. Table 1 shows that **PAC** achieves the best average score for both model sizes, with improvements over SEC of 13.1% on Qwen2.5-3B and 8.5% on Qwen2.5-7B. These gains indicate that **PAC** improves cross-difficulty transfer rather than only optimizing one validation split. SEC and DUMP remain competitive on some individual splits, suggesting that advantage-based curricula can identify locally useful update signals. However, their strengths are less consistent across model sizes and task types: high-advantage splits may continue to receive budget after reward-derived progress slows, whereas **PAC** shifts budget toward splits with active gains.

Multi-domain results show stronger cross-domain balance. Table 2 shows that **PAC** achieves the best curriculum average for both Qwen2.5-7B and Qwen2.5-32B, with improvements over DUMP of 6.2% and 1.9%. DUMP and Random remain strong on selected math benchmarks, but these localized gains do not translate into the best

Method	Countdown	Zebra	ARC	Avg.
Qwen2.5-3B				
Random	0.285 ± 0.021	0.251 ± 0.016	0.229 ± 0.013	0.255 ± 0.010
SEC	0.271 ± 0.019	$\underline{0.288} \pm 0.015$	$\underline{0.240} \pm 0.012$	$\underline{0.267} \pm 0.009$
DUMP	$\underline{0.301} \pm 0.017$	0.263 ± 0.014	0.202 ± 0.011	0.255 ± 0.009
Ours	$\mathbf{0.332} \pm 0.022$	$\mathbf{0.303} \pm 0.019$	$\mathbf{0.271} \pm 0.015$	$\mathbf{0.302} \pm 0.011$
ST	0.441 ± 0.024	0.328 ± 0.017	0.306 ± 0.014	0.358 ± 0.011
Qwen2.5-7B				
Random	0.393 ± 0.023	0.294 ± 0.017	0.306 ± 0.016	0.331 ± 0.011
SEC	$\underline{0.404} \pm 0.022$	$\mathbf{0.304} \pm 0.015$	0.313 ± 0.014	$\underline{0.340} \pm 0.010$
DUMP	0.375 ± 0.021	0.294 ± 0.018	$\underline{0.330} \pm 0.015$	0.333 ± 0.010
Ours	$\mathbf{0.439} \pm 0.026$	$\underline{0.296} \pm 0.013$	$\mathbf{0.373} \pm 0.019$	$\mathbf{0.369} \pm 0.012$
ST	0.487 ± 0.025	0.320 ± 0.016	0.417 ± 0.018	0.408 ± 0.012

Table 1: **Final validation performance on the multi-level reasoning setting after 500 training steps.** All results are reported as mean $\pm$ standard deviation over three seeds.

Method	Math			Code	Logic	Avg.
	MATH500	AMC22–23	AIME24	BigCodeBench	K&K	
Qwen2.5-7B						
Random	0.727 ± 0.020	$\mathbf{0.489} \pm 0.038$	0.133 ± 0.033	$\underline{0.217} \pm 0.013$	0.640 ± 0.016	0.435 ± 0.014
SEC	0.535 ± 0.021	0.387 ± 0.036	$\underline{0.135} \pm 0.035$	0.214 ± 0.012	0.851 ± 0.012	0.472 ± 0.013
DUMP	$\mathbf{0.745} \pm 0.018$	0.288 ± 0.034	0.108 ± 0.031	0.208 ± 0.014	$\underline{0.873} \pm 0.015$	$\underline{0.487} \pm 0.015$
Ours	$\underline{0.734} \pm 0.022$	$\underline{0.442} \pm 0.043$	$\mathbf{0.163} \pm 0.039$	$\mathbf{0.218} \pm 0.011$	$\mathbf{0.888} \pm 0.011$	$\mathbf{0.517} \pm 0.013$
ST	0.758 ± 0.017	0.518 ± 0.041	0.267 ± 0.036	0.250 ± 0.014	0.931 ± 0.010	0.565 ± 0.012
Qwen2.5-32B						
Random	0.648 ± 0.018	0.598 ± 0.040	0.246 ± 0.034	0.247 ± 0.013	0.923 ± 0.011	0.556 ± 0.013
SEC	0.820 ± 0.016	$\underline{0.605} \pm 0.038$	0.211 ± 0.031	0.236 ± 0.011	0.952 ± 0.009	0.578 ± 0.012
DUMP	$\mathbf{0.833} \pm 0.015$	0.598 ± 0.037	$\underline{0.247} \pm 0.032$	$\underline{0.250} \pm 0.015$	$\underline{0.957} \pm 0.008$	$\underline{0.589} \pm 0.014$
Ours	$\underline{0.822} \pm 0.019$	$\mathbf{0.625} \pm 0.044$	$\mathbf{0.259} \pm 0.028$	$\mathbf{0.253} \pm 0.010$	$\mathbf{0.977} \pm 0.012$	$\mathbf{0.600} \pm 0.011$
ST	0.840 ± 0.014	0.711 ± 0.042	0.400 ± 0.035	0.280 ± 0.012	0.994 ± 0.007	0.641 ± 0.011

Table 2: **Final validation performance on the multi-domain reasoning setting after 400 training steps.** All results are reported as mean $\pm$ standard deviation over three seeds. "Avg." first averages the three math benchmarks and then averages Math, Code, and Logic.

cross-domain average. By contrast, **PAC** maintains stronger average performance while improving on harder math benchmarks as well as on code and logic benchmarks, suggesting that the controller avoids overcommitting rollout budget to a single domain. Appendix D further analyzes the training-time curriculum dynamics in the multi-domain setting and shows how **PAC** reallocates rollout budget as reward-derived progress changes. Overall, these results suggest that effective curriculum allocation should jointly track where advantages remain large and where recent updates continue to yield measurable reward gains.

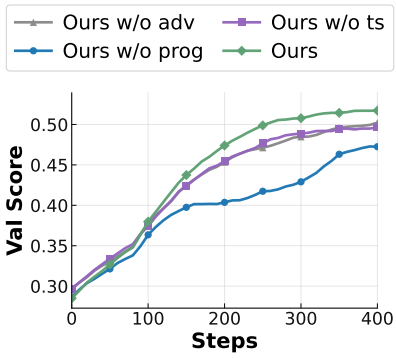


Figure 3: **Ablation of PAC on the multi-domain reasoning setting with Qwen2.5-7B.** "Val Score" denotes the mean of the task-level validation scores.

4.3 Ablation Study

We ablate **PAC**'s three design choices on the multi-domain reasoning setting with Qwen2.5-7B: **Ours w/o adv** sets $u_i^{(t)} = s_i^{(\text{prog})}$, **Ours w/o prog** sets $u_i^{(t)} = s_i^{(\text{adv})}$, and **Ours w/o ts** keeps the fused utility but replaces the Bayesian controller with Boltzmann Sampling over $u_i^{(t)}$. Appendix C formalizes each variant and reports the per-task breakdown. History-window sensitivity is reported in Appendix B.5.

All three components contribute, with reward-

derived progress the dominant factor: **PAC** improves over Ours w/o prog by 9.5% on the multi-domain average. This gap arises because the controller collapses to advantage-only allocation and keeps investing in arms whose advantages remain large after their rewards have saturated, the K&K Logic over-allocation visualized in Appendix D. The improvement over Ours w/o adv is 3.0%: a short-window reward slope alone is unreliable on slow-progress benchmarks such as AIME24 and BigCodeBench, where the learnability term separates improvement from noisy slopes. The improvement over Ours w/o ts is 4.2%: the fused utility is preserved but posterior uncertainty is discarded, so once a few arms develop higher utility estimates the allocation concentrates on them quickly.

5 Conclusion

In this work, we study online curriculum learning for multi-task RL post-training of LLMs, where rollout budgets must be allocated across heterogeneous tasks whose utility shifts during training. Advantage-only curricula capture the induced GRPO update magnitude but ignore whether it translates into reward gains. We therefore propose **PAC**, a Progress-Augmented Advantage Curriculum that estimates task utility by combining advantage-derived learnability with reward-derived progress, and uses Bayesian Thompson Sampling to adapt task allocation under uncertainty. Across multi-level and multi-domain reasoning settings, **PAC** reaches baseline-level validation averages with fewer rollout steps than random and advantage-based baselines. **PAC** improves final performance, achieving higher validation averages across model sizes and settings. These findings suggest that adaptive RL post-training curricula benefit from tracking both update potential and reward conversion.

Limitations

Reward signal assumptions. **PAC** is validated under RL post-training with outcome-level verifiable rewards, including rule-based answer checkers, symbolic verifiers, and unit-test execution. This setting matches many reasoning-oriented GRPO and RLVR pipelines and provides relatively dense reward observations from which window-level mean rewards can be reliably computed. Under sparser, noisier, or longer-horizon reward signals, such as those from learned reward models or process-level rewards, the linear trend estimator in Eq. (6) may need to be replaced by a more robust progress estimator that is less sensitive to reward noise and non-monotonic learning dynamics.

Task-level granularity. **PAC** maintains a Gaussian posterior per task arm, so the curriculum granularity is tied to the available task split structure, such as domain labels or difficulty buckets. When a single arm contains substantial internal diversity, for example, a math arm spanning a wide range of prompt difficulties, task-level allocation cannot exploit the variation within that arm. Extending **PAC** from task-level to prompt-level or instance-level sampling remains open and would require posterior structures that scale to arm counts far larger than the three and nine arms studied here.

Empirical scope. Our experiments evaluate **PAC** with backbones up to 32B parameters on reasoning, code, and logic mixtures with at most nine task arms. We did not directly study **PAC** on other capability families such as instruction following, dialogue, multilingual, or safety, on substantially larger arm sets, or under GRPO variants with alternative advantage normalization methods. We expect the multiplicative fusion in Eq. (8) and the Thompson Sampling controller in Section 3.3 to extend to these settings, but verifying this empirically remains future work.

References

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, JeongYeon Nam, and Donghyun Kwak. 2026. Online difficulty filtering for reasoning oriented reinforcement learning. *Preprint*, arXiv:2504.03380.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, ACM International Conference Proceeding Series, pages 41–48. ACM.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025. Self-evolving curriculum for llm reasoning. *Preprint*, arXiv:2505.14970.

Myungsik Cho, Jongeui Park, Suyoung Lee, and Youngchul Sung. 2024. Hard tasks first: Multi-task reinforcement learning through task scheduling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 8556–8577. PMLR.

François Chollet. 2019. On the measure of intelligence. *Preprint*, arXiv:1911.01547.

Thomas Foster, Anya Sims, Johannes Forkel, and Jakob Foerster. 2025. Lilo: Learning to reason at the frontier of learnability. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 48941–48974. Curran Associates, Inc.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. 2017. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Proceedings of Machine Learning Research, pages 1311–1320. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nat.*, 645(8081):633–638.

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

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. 2021. Prioritized level replay. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, pages 4940–4950. PMLR.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Derek Li, Jiaming Zhou, Leo Maxime Brunswic, Abbas Ghaddar, Qianyi Sun, Liheng Ma, Yu Luo, Dong Li, Mark Coates, Jianye Hao, and Yingxue Zhang. 2025. Omni-thinker: Scaling multi-task rl in llms with hybrid reward and task scheduling. *Preprint*, arXiv:2507.14783.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

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

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pages 22631–22648. PMLR.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, and Dongmei Zhang. 2025. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *Preprint*, arXiv:2308.09583.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2020. Teacher-student curriculum learning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9):3732–3740.

Dilxat Muhtar, Jiashun Liu, Wei Gao, Weixun Wang, Shaopan Xiong, Ju Huang, Siran Yang, Wenbo Su, Jiamang Wang, Ling Pan, and Bo Zheng. 2026. Complementary rl: Towards efficient experience-driven agent learning. *Preprint*, arXiv:2603.17621.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50.

OpenAI. 2024. Openai o1 system card. *CoRR*, abs/2412.16720.

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2026. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. *Preprint*, arXiv:2506.06632.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. 2020. Automatic curriculum learning for deep RL: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4819–4825. ijcai.org.

Yun Qu, Qi Wang, Yixiu Mao, Vincent Tao Hu, Björn Ommer, and Xiangyang Ji. 2026. Can prompt difficulty be online predicted for accelerating rl finetuning of reasoning models? *Preprint*, arXiv:2507.04632.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, and 21 others. 2022. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Qianli Shen, Daoyuan Chen, Yilun Huang, Zhenqing Ling, Yaliang Li, Bolin Ding, and Jingren Zhou. 2026. Bots: A unified framework for bayesian online task selection in llm reinforcement finetuning. *Preprint*, arXiv:2510.26374.

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. 2026. Efficient reinforcement finetuning via adaptive curriculum learning. *Preprint*, arXiv:2504.05520.

Mingyang Song, Mao Zheng, Zheng Li, Wenjie Yang, Xuan Luo, Yue Pan, and Feng Zhang. 2025. Fastcurl: Curriculum reinforcement learning with stage-wise context scaling for efficient training r1-like reasoning models. *Preprint*, arXiv:2503.17287.

Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. 2025. Improving data efficiency for llm reinforcement fine-tuning through difficulty-targeted online data selection and rollout replay. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 173449–173476. Curran Associates, Inc.

Yee Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. 2017. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, and 16 others. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhenting Wang, Guofeng Cui, Yu-Jhe Li, Kun Wan, and Wentian Zhao. 2025. Dump: Automated distribution-level curriculum learning for rl-based llm post-training. *Preprint*, arXiv:2504.09710.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pages 30745–30766.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, and 24 others. 2025b. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Yuanqiang Yu, Tianpei Yang, Yongliang Lv, Yan Zheng, and Jianye Hao. 2023. T3S: improving multi-task reinforcement learning with task-specific feature selector and scheduler. In *International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023*, pages 1–8. IEEE.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488. Curran Associates, Inc.

Yongcheng Zeng, Zexu Sun, Bokai Ji, Erxue Min, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Haifeng Zhang, Xu Chen, and Jun Wang. 2026. Cures: From gradient analysis to efficient curriculum learning for reasoning llms. *Preprint*, arXiv:2510.01037.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong

Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025. Group sequence policy optimization. *Preprint*, arXiv:2507.18071.

Erle Zhu, Dazhi Jiang, Yuan Wang, Xujun Li, Jiale Cheng, Yuxian Gu, Yilin Niu, Aohan Zeng, Jie Tang, Minlie Huang, and Hongning Wang. 2026. Data-efficient rlvr via off-policy influence guidance. *Preprint*, arXiv:2510.26491.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *Preprint*, arXiv:2406.15877.

A Advantage-Derived Learnability Derivation

A.1 Statement and Assumptions

Theorem A.1 (Absolute advantage as a learnability proxy)**.** *Fix the current policy π_θ and a task dataset $\mathcal{D}_i$. For prompts $x \sim \mathcal{D}_i$ and rollouts $\{o_j\}_{j=1}^G \sim \pi_\theta(\cdot \mid x)$, define*

$$L_i(\theta) = \mathbb{E}_{x\sim\mathcal{D}_i,\, o_{1:G}\sim\pi_\theta(\cdot|x)} \left[\frac{1}{G} \sum_{j=1}^{G} |\hat{A}_j| \right].$$

When the score-gradient norms $\|\nabla_\theta \log \pi_\theta(o_j \mid x)\|$ are bounded and vary smoothly across tasks, $L_i(\theta)$ tracks the scale of the GRPO update induced by task i up to slowly varying factors. Thus the mean absolute advantage provides an observable proxy for the learnability factor $\textbf{Learnability}_i$ *used in Section 3.2.1.*

A.2 Proof and Online Estimator

Proof. For a prompt x and its group of sampled responses $\{o_j\}_{j=1}^G$, the group-normalized advantage is

$$\hat{A}_j = \frac{r_j - \mathrm{mean}(\{r_1, \ldots, r_G\})}{\mathrm{std}(\{r_1, \ldots, r_G\}) + \epsilon},$$

matching the definition in Section 3.1. The policy gradient under common policy-gradient methods (e.g., PPO or GRPO) can be written as

$$g_x(\theta) = \mathbb{E}_{o_{1:G}\sim\pi_\theta(\cdot|x)} \left[\frac{1}{G} \sum_{j=1}^{G} \hat{A}_j \, \nabla_\theta \log \pi_\theta(o_j \mid x) \right]. \tag{A.1}$$

Aggregating over prompts from task i gives the task-conditioned gradient $g_i(\theta) = \mathbb{E}_{x\sim\mathcal{D}_i}[g_x(\theta)]$. Its scale is controlled by the absolute advantage and the score-gradient norm:

$$\|g_i(\theta)\|_2 \lesssim \mathbb{E}_{x\sim\mathcal{D}_i,\, o_{1:G}\sim\pi_\theta(\cdot|x)} \left[\frac{1}{G} \sum_{j=1}^{G} \left|\hat{A}_j\right| \, \|\nabla_\theta \log \pi_\theta(o_j \mid x)\|_2 \right]. \tag{A.2}$$

The right-hand side separates the task-conditioned update scale into two factors: the absolute advantage $|\hat{A}_j|$ and the score-gradient norm $\|\nabla_\theta \log \pi_\theta(o_j \mid x)\|_2$. Under the smoothness assumption in the theorem statement, the score-gradient norm changes more slowly across tasks than the reward separation captured by the normalized advantages. The main task-dependent term controlling the GRPO update scale is therefore the mean absolute advantage $L_i(\theta)$. Over the history window, this quantity is estimated by

$$a_i^{(t')} = \frac{1}{|\mathcal{X}_i^{(t')}|} \sum_{x\in\mathcal{X}_i^{(t')}} \frac{1}{G} \sum_{j=1}^{G} |\hat{A}_{t',x,j}|,$$

$$s_i^{(\mathrm{adv})} = \frac{1}{W} \sum_{t'=t-W}^{t-1} a_i^{(t')}.$$

which is the advantage-derived learnability score in Eq. (5). Thus $s_i^{(\mathrm{adv})}$ estimates $\textbf{Learnability}_i$: the amount of advantage-driven update signal that task i can still provide under the current policy. □

B Implementation Details

B.1 Data, Arms, and Verifiers

Table 3 summarizes the arm definitions, data sizes, validation splits, and automatic reward functions for the two experimental settings.

B.2 Training and Sampling

Training configuration. All experiments are run on a single node with 8 NVIDIA GPUs. At each training step, the curriculum sampler first constructs a batch with size $B = 256$ in both the multi-level and multi-domain settings. For each selected prompt, the current policy samples $G = 8$ responses, which are scored by the corresponding verifiable reward function. Group-normalized advantages are computed only within the G responses generated for the same prompt, so reward scales are not normalized across different tasks or domains before the task-level statistics are accumulated. The actor optimizer uses a PPO mini-batch size of 64. Prompts are truncated to 2,048 tokens in both the multi-level and multi-domain settings; generated responses are truncated to 4,096 tokens in both settings. Within each experimental setting, these GRPO hyperparameters are kept fixed across model sizes, curriculum baselines, and ablation variants.

Task arms and sampling. The curriculum controller operates over the training splits described in Section 4.1; all multi-task methods receive the same arm set in a given experimental setting. When an arm is selected, we draw one prompt uniformly from the corresponding training split. For the multi-level setting, the arms follow the available training difficulty splits. For the multi-domain setting, the arms follow the domain-specific training splits used to form the math, code, and logic mixture. For all multi-task runs, every prompt is drawn through the corresponding curriculum sampler. Table 3 summarizes the task arms, data sizes, and reward functions used in the two experimental settings. In the multi-domain setting, the source corpora naturally provide different numbers of filtered, verifier-compatible examples. We retain each filtered corpus as the prompt pool for its domain; per-step domain exposure is determined by the arm-level sampler, so corpus size affects only which prompts can be drawn after an arm is selected.

B.3 Controller State and Utility Estimation

PAC state updates. We initialize each arm in PAC with a Gaussian posterior mean $\mu_i = 0$ and variance $\sigma_i^2 = 1$. The cold-start length is fixed to $T_{\text{cold}} = 50$ steps in both the multi-level and multi-domain settings. During this stage, batch construction is uniform, but the controller still records rewards and advantages and updates the posterior, giving each arm a comparable amount of initial evidence. After the cold-start stage, batches are formed by the sequential Thompson Sampling procedure in Section 3.3: for each training sample, the controller draws one utility value from every arm posterior and assigns the sample to the arm with the largest draw. For each arm, the resulting count $c_i^{(t)}$ determines the evidence weight $\rho_i^{(t)} = c_i^{(t)}/(B/N)$ in Eq. (10). After each posterior update, we apply the non-stationary variance inflation in Eq. (11) to preserve exploration as task utilities change during training. We use inflation variance $\delta^2 = 0.02$ and set all numerical stability constants ϵ in the advantage normalization and Eq. (7) to 10^{-9} for all experiments.

Utility estimation. We compute the advantage-derived learnability score $s_i^{(\text{adv})}$ as the average step-level mean absolute group-normalized advantage over prompts from arm i in the history window, and the reward-derived progress estimate in Eq. (6) uses the same window over the step-level mean reward of each arm. When an arm has too few recent observations for a reliable trend fit, we set its fitted slope to zero before cross-task normalization, so that the PAC controller relies only on training-time rewards, advantages, and sampling counts, with validation scores reserved for reporting. The window length W sets the time scale of these estimates: a short window is overly sensitive to individual rollout batches, whereas a long window lags behind saturation or delayed improvement. We use $W = 16$ in all experiments, which balances smoothing single-batch reward noise against tracking non-stationary changes in task utility; the trend fit, cross-task slope normalization, and zero-slope fallback further reduce sensitivity to the exact window length. Combined with the cold-start length above, this single window choice keeps the controller on a consistent time scale across settings, rather than tuning curriculum hyperparameters to validation benchmarks.

B.4 Validation Performance at Individual Difficulty Levels

The main multi-level evaluation in Table 1 focuses on the extremely hard split to measure transfer be-

Setting	Training arms	Data size and validation	Reward / verifier
Multi-level	9 arms from {Countdown, Zebra, ARC} × difficulty levels 1–3, defined by `extra_info.type` and `extra_info.difficulty`.	90,000 training examples in total, with 10,000 examples per arm. Validation files contain 800 examples per task, with 200 examples for each difficulty level 1–4; reported out-of-distribution validation uses level 4.	Countdown checks whether the boxed expression uses each number exactly once and reaches the target. Zebra and ARC compare the boxed answer against the rule-based target.
Multi-domain	3 arms from {Math, Code, K&K Logic}, defined by `extra_info.type`. K&K 3-, 4-, and 5-character splits are grouped into the logic arm.	26,657 training examples: 11,499 from DAPO-Math, 12,458 from Code-R1, and 2,700 from K&K Logic. Validation uses 500 MATH500 examples, 83 AMC examples, 30 AIME examples, BigCodeBench for code, and 100 K&K 6-character examples.	Math uses the DeepScaleR-style symbolic and numeric answer verifier. Code uses unit-test execution for Code-R1 and BigCodeBench tasks. K&K uses an answer verifier for knight/knave assignments.

Table 3: **Dataset, arm, and reward statistics for the two experimental settings.**

yond the training difficulty range. To test whether the improvement is confined to that split, Table 4 reports final scores on held-out validation portions at all four difficulty levels, averaged across Countdown, Zebra, and ARC. All validation examples are disjoint from the RL training data, including the simple, medium, and hard levels used during training.

Method	Simple	Medium	Hard	Ext.
Random	0.637	0.479	0.409	0.255
SEC	0.643	0.488	0.413	0.267
DUMP	0.732	**0.558**	0.412	0.255
Ours	**0.734**	0.552	**0.466**	**0.302**

Table 4: **Validation performance at individual difficulty levels in the multi-level setting.** Scores are measured after 500 training steps with Qwen2.5-3B and averaged across Countdown, Zebra, and ARC. "Ext." denotes the extremely hard level.

Table 4 shows that **PAC** preserves performance at lower difficulty while its largest gains emerge at higher difficulty. Relative to the strongest baseline in each column, **PAC** is within 1.1% on Simple and Medium, while improving Hard and Ext. by 12.8% and 13.1%, respectively. This pattern indicates that the adaptive curriculum does not trade away performance on easier levels for out-of-distribution transfer; instead, it yields its clearest benefit where the reasoning problems are more difficult.

B.5 History-Window Sensitivity

We examine the sensitivity of **PAC** to the history-window length W, which determines the time scale of advantage smoothing in Eq. (5) and reward-trend estimation in Eq. (6). We evaluate six window lengths, $W \in \{4, 16, 32, 64, 128, 256\}$, in the nine-arm multi-level setting with Qwen2.5-3B, holding all other settings fixed.

W	Countdown	Zebra	ARC	Avg.
4	0.245	0.240	0.228	0.238
16	0.332	0.303	0.271	0.302
32	0.326	0.298	0.264	0.296
64	0.288	0.271	0.253	0.271
128	0.252	0.248	0.232	0.244
256	0.248	0.244	0.227	0.240

Table 5: **Sensitivity to the history-window length W.** We report validation accuracy after 500 training steps on the held-out extremely hard splits in the multi-level setting with Qwen2.5-3B. $W = 16$ is the default.

Table 5 reveals a trade-off between estimation noise and responsiveness. **An overly short window is noisy.** With $W = 4$, the average drops by 21.2% relative to the default (0.238 vs. 0.302). This window averages the advantage signal over only four recent steps and fits the reward slope from only four task-level reward observations. Because individual mixed batches can have noisy task rewards and few samples for some arms (Section 3.2), the controller can respond to transient fluctuations rather than sustained learning progress.

An overly long window adapts too slowly. Increasing W to 64, 128, and 256 reduces the average by 10.3%, 19.2%, and 20.5%, respectively, relative to $W = 16$. Since task utility is non-stationary, a long window retains observations generated by substantially earlier policies. Both the smoothed advantage estimate and the fitted reward trend can therefore lag behind task saturation or newly emerging progress, delaying the reallocation that **PAC** is

designed to perform.

PAC is stable near the default. $W = 32$ reaches 0.296, only 2.0% below the default 0.302, with similarly small differences on all three tasks. Thus, $W = 16$ provides the best balance between smoothing rollout noise and tracking changing task utility, while the result at $W = 32$ shows that PAC is not narrowly tuned to a single window value.

C Ablation Study Details

This appendix formalizes the three ablation variants used in Section 4.3 and reports their per-task validation performance. All variants share the same multi-domain training data, cold-start length T_{cold}, history window W, and GRPO hyperparameters as the full PAC configuration. They differ from the full model in exactly one place: either the utility observation that feeds the controller, or the rule used to convert utilities into rollout allocation.

C.1 Variant Definitions

Utility ablations: *Ours w/o adv* and *Ours w/o prog*. The two utility ablations leave the Gaussian posterior in Eq. (9) and the Thompson Sampling controller of Section 3.3 unchanged, and modify only the utility observation that drives the posterior update. Recall from Eq. (8) that the full model uses the multiplicative fusion

$$u_i^{(t)} = \left(1 + s_i^{(\text{prog})}\right) \cdot s_i^{(\text{adv})},$$

in which $s_i^{(\text{adv})}$ estimates the learnability factor $\textbf{Learnability}_i$ and $1 + s_i^{(\text{prog})}$ estimates the conversion factor $\textbf{Convert}_i$ in the factorization of Eq. (4). The two utility ablations isolate each factor:

$$u_{i,\text{w/o adv}}^{(t)} = s_i^{(\text{prog})}, \tag{C.1}$$

$$u_{i,\text{w/o prog}}^{(t)} = s_i^{(\text{adv})}. \tag{C.2}$$

Eq. (C.1) drops the learnability factor and lets reward-derived progress drive allocation on its own. Eq. (C.2) drops the conversion factor and recovers an advantage-only utility, which is the same type of signal used by SEC and DUMP.

Controller ablation: *Ours w/o ts*. The controller ablation leaves the fused utility of Eq. (8) unchanged and replaces only the allocation rule. Concretely, **Ours w/o ts** discards the Gaussian posterior of Eq. (9) and assigns each prompt in the rollout batch to arm i according to a Boltzmann distribution over the current utility scores,

$$p_i^{(t)} = \frac{\exp\left(u_i^{(t)}\right)}{\sum_{j=1}^{N} \exp\left(u_j^{(t)}\right)}. \tag{C.3}$$

Under this rule, the controller does not model its own uncertainty: the precision-weighted posterior update in Eq. (10) is unused, and the variance inflation in Eq. (11) is inactive. Arms with larger utilities receive higher sampling probabilities, but the allocation is deterministic given the utility scores and cannot trade exploitation against posterior variance.

C.2 Per-Task Results

Table 6 reports the final validation accuracy of each variant on the five multi-domain benchmarks. Three patterns are consistent with the analysis in Section 4.3. First, **Ours w/o prog** loses most ground on MATH500 and AMC22–23, where it undersamples the Math arm because the saturating K&K arm continues to dominate an advantage-only utility; this behavior is analyzed in Appendix D. Second, **Ours w/o adv** retains the progress signal and stays close to the full model on the easier benchmarks, but loses accuracy on AIME24 and BigCodeBench, the two splits where reward gains are slow and the progress estimator is noisier; the learnability term helps distinguish slow reward gains from noisy slopes in this regime. Third, **Ours w/o ts** keeps the fused utility but records the lowest K&K accuracy among the variants, because Boltzmann Sampling collapses the K&K share once Math and Code utilities exceed it, and the resulting rule has no analogue of variance inflation to revisit the arm.

D Analysis of Curriculum Dynamics in Multi-Domain Training

Figure 4 exposes the controller-internal trajectory of PAC during multi-domain training of Qwen2.5-7B: per-arm sampling probability, advantage-derived learnability $s_i^{(\text{adv})}$, and reward-derived progress $s_i^{(\text{prog})}$, each averaged within a 50-step plotting window. Reading the figure left to right shows how the rollout budget moves between arms; reading it across the two utility columns shows which signal is causing that movement. We use the figure for two purposes: to verify that PAC discovers a non-trivial curriculum once Thompson

Method	Math			Code	Logic	Avg.
	MATH500	AMC22–23	AIME24	BigCodeBench	K&K	
Ours w/o adv	0.721	**0.462**	0.128	0.213	0.856	0.502
Ours w/o prog	0.535	0.387	0.135	0.214	0.851	0.472
Ours w/o ts	0.726	0.450	0.141	0.213	0.837	0.496
Ours	**0.734**	0.442	**0.163**	**0.218**	**0.888**	**0.517**

Table 6: **Per-task ablation results on the multi-domain reasoning setting with Qwen2.5-7B after 400 training steps.** "Avg." first averages the three math benchmarks and then takes the mean over Math, Code, and Logic.

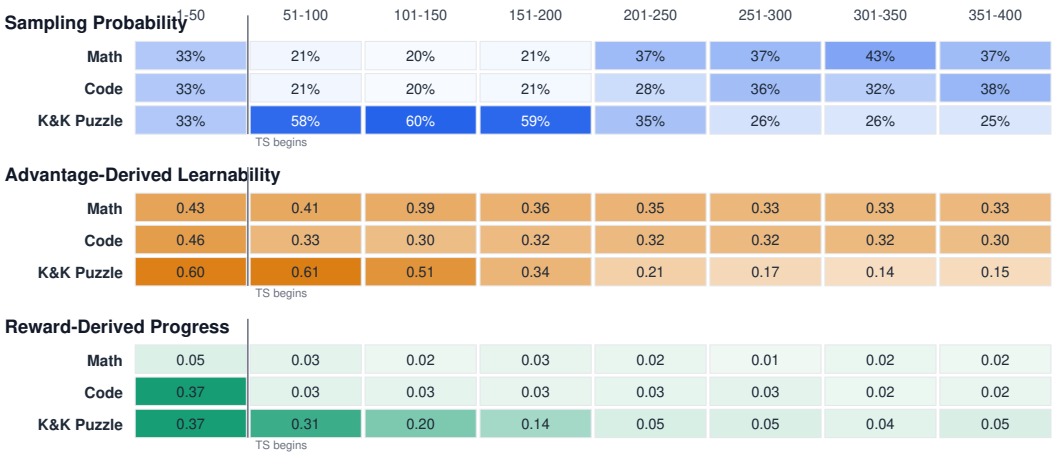


Figure 4: **Training-time curriculum dynamics of PAC on the multi-domain setting with Qwen2.5-7B.** Columns report, for each task arm, the sampling probability, the advantage-derived learnability $s_i^{(\mathrm{adv})}$, and the reward-derived progress $s_i^{(\mathrm{prog})}$; rows correspond to the Math, Code, and K&K Logic arms. Each cell aggregates the mean over a 50-step non-overlapping plotting window. The vertical marker indicates the end of the uniform cold-start stage at step 50, after which Thompson Sampling becomes the allocation rule.

Sampling activates at step 50, and **to attribute the multi-domain validation gain in Table 2 to a concrete reallocation mechanism** rather than to unattributed optimizer stochasticity.

D.1 Three-Phase Allocation Trajectory

The sampling-probability column separates into three phases. The cold-start window allocates roughly a third of the rollout budget to each arm, so every arm accumulates comparable initial evidence before the posterior update in Eq. (10) starts to differentiate them. Once Thompson Sampling becomes active, the controller concentrates budget on K&K Logic, whose share rises to about 60% within roughly the next 100 training steps; during this phase K&K Logic records both the highest advantage-derived learnability and the highest reward-derived progress, so the fused utility in Eq. (8) ranks it first under both signals simultaneously. After step 200, **the budget rotates back toward Math and Code, and the K&K Logic share settles near one-quarter of the batch** for the remainder of training.

D.2 Reward-Derived Progress Drives the Reallocation

The K&K Logic arm reveals why a controller needs more than advantage magnitude. Its advantage-derived learnability $s_i^{(\mathrm{adv})}$ decreases by roughly half over training but remains comparable to the other two arms, so an advantage-only scheduler would continue to over-allocate rollouts to K&K Logic. The reward-derived progress signal $s_i^{(\mathrm{prog})}$ decays much more rapidly as the policy approaches the K&K Logic reward ceiling, because further updates no longer convert into measurable reward gains. The multiplicative fusion in Eq. (8) couples the two signals, so **the collapse of progress pulls the fused utility down even while learnability remains nominally healthy**, and the controller redirects rollouts toward arms where additional updates still translate into reward. This is the failure mode that motivates the conversion factor $\mathbf{Convert}_i$ in Section 3.2, and Figure 4 shows that PAC corrects

it online, while training is still in progress, rather than only in retrospect.

D.3 A Gradual Rather Than Abrupt Transition

The Math and Code arms gain budget through the same factorization but with a different signal mix. For Math, both learnability and progress decay only slowly throughout training; for Code, the progress signal strengthens once its rollouts reliably pass unit-test verification. Both arms then offer a more favorable fused utility than the saturating K&K Logic arm, and Thompson Sampling raises their posterior means accordingly. The transition is gradual rather than abrupt because **the variance inflation in Eq. (11) preserves posterior uncertainty and prevents premature exploitation** of the arm with the largest current utility estimate. The same mechanism explains why the K&K Logic share stabilizes near a quarter of the batch instead of collapsing to zero: with inflated variance, the controller keeps K&K Logic available for periodic resampling, so that it can still detect any later utility shift on that arm.

D.4 From Reallocation to the Multi-Domain Validation Gain

The mid-training reallocation provides a mechanistic account of the multi-domain trends in Section 4.2. The **PAC** validation curve in Figure 2b begins to separate from DUMP after the mid-training phase, and this separation coincides with the window in which the controller redirects rollout budget away from the saturating K&K Logic arm and toward Math and Code. Reward-derived progress is therefore the decisive signal in the regime where advantage magnitude alone would mislead the controller, and **the resulting reallocation accounts for the 6.2% cross-domain average improvement of PAC over advantage-only baselines** reported for Qwen2.5-7B in Table 2.

E Prompt Examples

This appendix shows representative prompts from the datasets used in the two experimental settings. To keep the examples readable, we include the task-specific user content and omit the shared conversation preamble and assistant cue. We also report the reward target for each example to clarify how automatic verification is performed; these targets are not part of the model input.

E.1 Setting 1: Multi-Level Reasoning

Countdown Prompt

Prompt:
Using the numbers $21, 95, 8, 2, 26, 2$ and basic arithmetic operations $(+, -, \times, /)$, create an expression that equals 30. You need to use each number exactly once. Present the final expression that equals the target number within `\boxed{}`. For example: `\boxed{46+68/(52-50)}`.
Reward target: The expression must use $21, 95, 8, 2, 26, 2$ exactly once and evaluate to 30.

Zebra Prompt

Prompt:
This is a logic puzzle. There are 4 houses, numbered 1 on the left to 4 on the right, from the perspective of someone standing across the street from them. Each house has a different person, and each person has different characteristics:

- Each person has a unique name: carol, arnold, alice, bob.
- Everyone has a favorite smoothie: butterscotch, desert, darkness, dragonfruit.
- They all have a different favorite flower: iris, daffodils, lilies, carnations.
- Everyone has something different for lunch: grilled cheese, stir fry, pizza, soup.

1. Bob is the person who loves stir fry.
2. The person who loves eating grilled cheese is directly left of the person who loves a carnations arrangement.
3. The person who loves a carnations arrangement and the person who loves a bouquet of daffodils are next to each other.
4. The person who loves the soup is directly left of the Butterscotch smoothie drinker.
5. The Darkness smoothie drinker is directly left of the Desert smoothie lover.
6. The person who loves the soup is the Desert smoothie lover.
7. The person who loves a carnations arrangement is Bob.
8. The person who loves the bouquet of lilies is the Butterscotch smoothie drinker.
9. The Desert smoothie lover is Carol.
10. The person who loves the bouquet of iris is Arnold.

What is the name of the person who lives in House 1? Provide only the name as the final answer and put it in `\boxed{}`.
Reward target: `arnold`.

ARC Prompt

Prompt:
Find the common rule that maps an input grid to an output grid, given the examples below.
Example 1:
Input: `0 0 0 9 3 6 0 0 0 0`
Output: `9 3 6 0 0 0 0 0 0 0`
Example 2:
Input: `0 0 0 0 0 0 0 0 2 8`
Output: `0 0 0 0 0 2 8 0 0 0`
Example 3:
Input: `0 0 0 0 0 0 0 9 0 0`
Output: `0 0 0 0 9 0 0 0 0 0`
Below is a test input grid. Predict the corresponding output grid by applying the rule you found. Describe how you derived the rule and your overall reasoning process before submitting the answer. The final answer must be placed in `\boxed{}` and should be just the test output grid itself.
Input:
`1 6 7 8 9 4 0 0 0 0`
Reward target: `8 9 4 0 0 0 0 1 6 7`.

E.2 Setting 2: Multi-Domain Reasoning

Math Prompt

Prompt:
Every morning Aya goes for a 9-kilometer-long walk and stops at a coffee shop afterwards. When she walks at a constant speed of s kilometers per hour, the walk takes her 4 hours, including t minutes spent in the coffee shop. When she walks $s+2$ kilometers per hour, the walk takes her 2 hours and 24 minutes, including t minutes spent in the coffee shop. Suppose Aya walks at $s+\frac{1}{2}$ kilometers per hour. Find the number of minutes the walk takes her, including the t minutes spent in the coffee shop. Put the final answer within `\boxed{}`.
Reward target: $\boxed{204}$.

Code Prompt

Prompt:
You are a helpful programming assistant. The reasoning process and answer are enclosed within `<think>...</think>` and `<answer>...</answer>` tags, respectively.
User: Generate a random string of the specified length composed of uppercase and lowercase letters, and then count the occurrence of each character in this string. The function should raise `ValueError` if the length is a negative number. The function should output:

> `dict`: a dictionary where each key is a character from the generated string and the value is the count of how many times that character appears in the string.

You should write self-contained code starting with:

```
import collections
import random
import string
def task_func(length=100):
```

Reward target: Unit tests validate the implementation of entry point `task_func`.

K&K Logic Prompt

Prompt:
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie. You meet 6 inhabitants: Grace, Penelope, Mia, Noah, Matthew, and Lily. Grace said, "Noah is a knave." Penelope asserted, "Mia is a knight." Mia said, "Grace is a knave." Noah said, "Mia is a knight if and only if Matthew is a knight." Matthew said that Noah is not a knave. Lily said, "Mia is not a knight." So who is a knight and who is a knave?
Reward target:

> (1) Grace is a knave
> (2) Penelope is a knight
> (3) Mia is a knight
> (4) Noah is a knight
> (5) Matthew is a knight
> (6) Lily is a knave