POLICY-MERGE: POLICY PREDICTION FOR MULTI-DOMAIN RL AND DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are post-trained with Reinforcement Learning (RL) across many domains, such as math, code, and tool use. The resulting model depends on how data and compute are allocated across these domains, whether by training on a data mixture or by multi-teacher distillation. Evaluating an allocation, however, requires a full training run. Existing mixture selection methods reduce this cost by predicting benchmark performance or using proxy models for mixture optimization, but do not directly predict the policy obtained by training on a mixture. We introduce a method that approximates the policy of a model trained with RL on a given data mixture by composing independently post-trained domain experts at decoding time. We compare this approximated policy with the model trained with RL on the mixture, and show that they perform similarly across benchmarks and are distributionally close. This allows for predicting the capabilities of a specified mixture. Our method enables efficient data mixture selection, as candidate mixtures can be evaluated and ranked in policy space before committing compute to post-training. Finally, we show that a student distilled on-policy from several experts at once to minimize a weighted sum of per-token KL divergences is equivalent to using our approximated policy as a teacher. One can thus skip the multi-teacher distillation and decode the approximated policy directly.

1 INTRODUCTION

Modern large language models are post-trained on many task domains such as coding, math, instruction following, and safety (Grattafiori et al., 2024; Yang et al., 2025). The composition of these domains, or how much data and compute we allocate to learning from each domain shapes the capabilities and behaviour of the final model (Team et al., 2025). The process of finding the optimal data mixture can be costly (Yang et al., 2025; 2026), or require trial-and-error. For example, Tulu 3 selects its post-training mixture by training skill specific mixtures and iteratively adding and removing datasets to improve performance (Lambert et al., 2024).

One may ask why more data is not always better, or why not simply gather all available data and train jointly (OLMo et al., 2025; Grattafiori et al., 2024; Team et al., 2025)? Performance in each domain scales at different rates under RL, performance improvement in some domains can saturate before others, so it is inefficient to use all data.

The difficulty in data mixture optimization is that the space of different data mixture allocations is combinatorially large (Hu et al., 2026). Additionally, choosing the allocation is unusually expensive in RL, as every training mixture requires a full on-policy run whose cost is dominated by rollout generation (Olmo et al., 2025). The problem is further exacerbated by the fact that post-training pipelines continuously incorporate new tasks, data sources, and RL environments (Zeng et al., 2026; Fang et al., 2026; Xiaomi MiMo Team, 2026). Each time a domain is added or updated, previously tuned allocations may no longer be optimal, and the search must be repeated. One solution is to avoid joint training altogether and instead train one specialized expert per domain using RL, then combine these experts into a single student through multi-teacher on-policy distillation (Team et al., 2026; DeepSeek-AI et al., 2026; Gao et al., 2026; Ma et al., 2026), where the teacher provides a probability distribution at every token of the student's trajectory (Xiao et al., 2026; Lu & Lab, 2025). However, factorizing training does not eliminate the allocation problem. Whether one trains on a joint data mixture or distills from a collection of experts, the final model still depends on how

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

much influence is assigned to each domain (Li et al., 2026). Evaluating a candidate allocation still requires either a full RL run or a full distillation run, and the combinatorial cost remains. Ideally, we would like to answer a counterfactual question before committing to a data mixture for an RL or on-policy distillation run:

Can we predict a policy trained on a particular mixture of domains?

A natural possibility is that the experts already reveal how unseen training data allocations would behave. Recent work has explored this idea in parameter space (Wang et al., 2026b; Hu et al., 2026), using combinations of independently trained domain experts as proxies for models trained on different data mixtures. Alternatively, we can combine the policy distributions instead. Prior work (Hamidieh et al., 2026) has shown that policies trained with different rewards are linearly connected in the policy log-ratio space. In that case, product-of-experts aggregation of the policy distributions (Heskes, 1997; Liu et al., 2021) can be useful to predict RL outcomes on new rewards.

Our setting focuses on predicting policies trained on a given data mixture, from experts trained on separate domains. We first show that combinations of domain expert policy log-ratios can approximate those of models trained on mixtures of these domains. Unlike reward composition, the requested data proportions do not directly specify the corresponding coefficients in policy space, since different domains can lead to policy drifts of substantially different magnitudes under the same training recipe. We then show that useful composition coefficients can be obtained *without* access to those models trained on data mixtures.

We propose POLICY-MERGE, which uses these coefficients to compose experts policy distributions. We show that it approximates the policy that would have been obtained by directly RL training on the specified allocation, before that training run is performed. We evaluate predictions through downstream performance and distributional similarity, and evaluate their usefulness for selecting data mixtures. We also establish a connection between policy composition and multi-teacher distillation by showing that weighted per-token distillation loss can be represented using a single composed teacher.

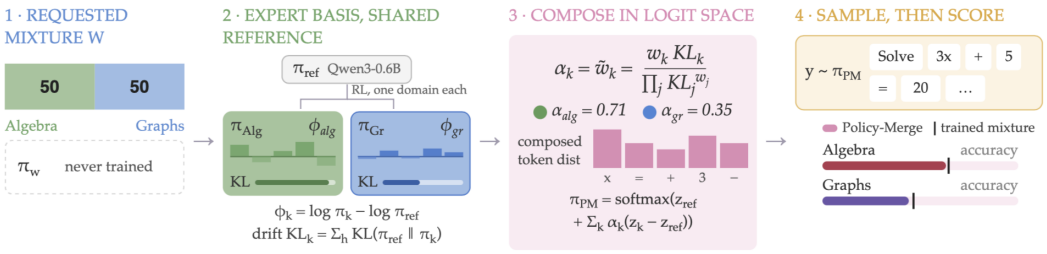


Figure 1: **Overview of POLICY-MERGE.** Given a data mixture $\mathbf{w}$, POLICY-MERGE approximates the model $\pi_{\mathbf{w}}$ trained with RL on $\mathbf{w}$ by sampling from a weighted logit composition (product) of domain experts.

Our contributions are as follows:

- We introduce POLICY-MERGE, a method for approximating the policy produced by multi-domain post-training by composing their next-token distributions directly in policy space.
- We show that POLICY-MERGE recovers the behavior of models actually trained on mixtures, in terms of both downstream performance and distributional similarity.
- We show how this approximation can be used to evaluate candidate data mixtures without training each one, which enables mixture selection into search over inexpensive policy compositions.
- We connect POLICY-MERGE to multi-teacher on-policy distillation, and show that distillation from a weighted collection of experts can be expressed as distillation from a single composed teacher policy.

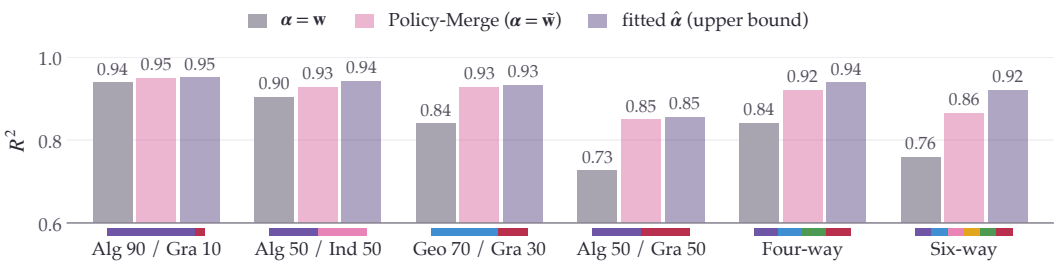


Figure 2: **Expert policy log-ratios linearly predict the policy of a model trained on a data mixture**: We fit the regression in Eq. (2) and find that the mixture policy can be expressed as a linear combination of expert log-ratios. We find using data proportions as coefficients ($\boldsymbol{\alpha} = \mathbf{w}$), is further from fitted ground-truth coefficients $\hat{\alpha}$, and proposed KL weighted coefficients from the POLICY-MERGE ($\boldsymbol{\alpha} = \tilde{\mathbf{w}}$, Section 2.3), are closer to performance in fitted weights. Details in Appendix B.3.

2 PREDICTING MIXTURE TRAINING OUTCOMES

We hypothesize that policies trained independently on individual domains can predict the outcome of training on a mixture of those domains. Concretely, we hypothesize that the policy log-ratio of a model trained on a data mixture is a linear combination of the experts' policy log-ratios. In this case, composing the experts at decoding time reproduces the mixture policy given the right coefficients (Section 2.2). Coefficients fitted to trained mixtures account for most of how far each moves from the reference policy (Figure 2). We predict the coefficients from the data proportions (Section 2.3), and use the composed policy as a teacher for on-policy distillation (Section 2.4).

2.1 PROBLEM SETUP

Let $\mathcal{D}_1, \ldots, \mathcal{D}_K$ be the prompt distributions of K training domains. We specify a data mixture by a vector $\mathbf{w} \in \Delta^{K-1}$, such that the mixture has proportion w_k of domain $\mathcal{D}_k$. Let π_{ref} be a reference model and let $\mathcal{B} = \{\pi_1, \ldots, \pi_K\}$ be a basis of policies obtained by post-training the same reference model on K different domains. Each basis policy π_k is trained with RL from π_{ref} on prompts from $\mathcal{D}_k$ with a verifiable reward $r \in [0, 1]$: the task verifier's score, 1 for a correct answer, with partial credit on some tasks.

Given a requested data mixture $\mathbf{w} \in \Delta^{K-1}$, a reference policy π_{ref}, and independently trained domain experts $\{\pi_k\}_{k=1}^K$, our goal is to predict composition coefficients $\boldsymbol{\alpha}$ from $\mathbf{w}$ and use them to construct a prediction $\widehat{\pi}_{\mathbf{w}}$ of the policy $\pi_{\mathbf{w}}$ that RL post-training π_{ref} on $\mathbf{w}$ with the same recipe would produce, without training or observing $\pi_{\mathbf{w}}$.

2.2 COMPOSING DOMAIN EXPERTS

For a prompt x and response $y = (y_1, \ldots, y_T)$, let $h_t = (x, y_{<t})$ be the prefix at step t and $v \in \mathcal{V}$ a candidate next token. The *policy log-ratio* of expert k,

$$\phi_k(h_t, v) = \log \pi_k(v \mid h_t) - \log \pi_{\text{ref}}(v \mid h_t), \tag{1}$$

measures how training on domain k changed the model's next-token distribution relative to π_{ref}. It is positive for tokens that the expert made more likely than π_{ref}. We define $\phi_{\mathbf{w}}$ in the same way for the mixture policy $\pi_{\mathbf{w}}$.

Prior work (Hamidieh et al., 2026) has shown that adding the weighted log-ratios of experts trained on different rewards to $\log \pi_{\text{ref}}$ predicts the policy obtained by RL on the same weighted combination of rewards, and that the experts' log-ratios across many rewards span an approximately low-rank space. This is derived for RL with a KL penalty to π_{ref}. For RLVR, experts can be trained without this KL penalty (Liu et al., 2025; Yu et al., 2026), as ours are. For data mixtures, we therefore hypothesize a similar property, which we call *linear composition*, and find that it holds empirically (Figure 2). Specifically, it assumes that training on a mixture changes the log-probability of each token by a weighted sum of the changes that the experts made, with one coefficient per expert, shared across all prefixes and tokens. For every mixture $\mathbf{w}$, there are coefficients $\boldsymbol{\alpha} \in \mathbb{R}^K$ such that

$$\phi_{\mathbf{w}}(h_t, v) = \sum_{k=1}^{K} \alpha_k \, \phi_k(h_t, v) - \log Z_{\boldsymbol{\alpha}}(h_t), \tag{2}$$

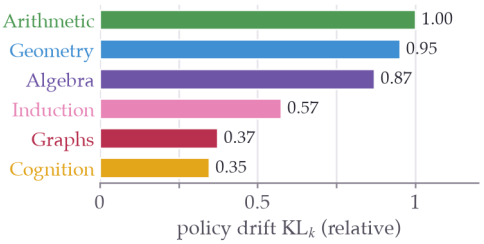


Figure 3: **Policy drift KL_k (Eq. (4)) across experts varies.** (relative to the largest), although all experts share same initialization and training recipe, some end up further away from the reference policy.

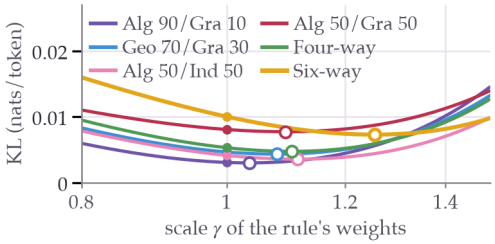


Figure 4: **$\tilde{\mathbf{w}}$ is close to KL-optimal weights for policy composition.** KL to each trained mixture when scaling $\tilde{\mathbf{w}}$ by γ. Dots ($\gamma = 1$) lie near the minima (rings).

where $Z_{\boldsymbol{\alpha}}(h_t) = \sum_{v'} \pi_{\mathrm{ref}}(v' \mid h_t) \exp\big(\sum_k \alpha_k \phi_k(h_t, v')\big)$ makes the probabilities at h_t sum to one. It depends only on the prefix, not on the token, so it does not change the relative probabilities of tokens.

This assumption lets us reproduce the mixture policy by composing the experts at decoding time. For coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K) \in \mathbb{R}^K$, define

$$\begin{aligned} \pi_{\mathrm{PM}(\boldsymbol{\alpha})}(v \mid h_t) &= \tfrac{1}{Z_{\boldsymbol{\alpha}}(h_t)} \pi_{\mathrm{ref}}(v \mid h_t) \exp\big(\textstyle\sum_{k=1}^{K} \alpha_k \, \phi_k(h_t, v)\big) \\ &= \tfrac{1}{Z_{\boldsymbol{\alpha}}(h_t)} \pi_{\mathrm{ref}}(v \mid h_t)^{1-\sum_k \alpha_k} \textstyle\prod_{k=1}^{K} \pi_k(v \mid h_t)^{\alpha_k}. \end{aligned} \tag{3}$$

Exponentiating Eq. (2) gives $\pi_{\mathbf{w}} = \pi_{\mathrm{PM}(\boldsymbol{\alpha})}$. So with the right coefficients, the composition is the mixture policy, obtained without training on the mixture.

Each expert increases or decreases the log-probability of every token by α_k times its own log-ratio. Setting $\boldsymbol{\alpha} = \mathbf{0}$ recovers π_{ref}, and setting $\alpha_k = 1$ with all other coefficients zero recovers π_k. The coefficients do not have to sum to one, so the weight on π_{ref}, $1 - \sum_k \alpha_k$, can be negative. In terms of next-token logits $z_{\mathrm{ref}}(h_t)$ and $z_k(h_t)$, Eq. (3) is $\mathrm{softmax}\big(z_{\mathrm{ref}} + \sum_k \alpha_k (z_k - z_{\mathrm{ref}})\big)$, so sampling needs only the next-token logits of the reference and the experts (Algorithm 1).

2.3 Composition Coefficients from Expert Policy Drift

Linear composition (Eq. (2)) shows that *some* coefficients reproduce $\pi_{\mathbf{w}}$, but not which ones. The simplest guess is the domain weights themselves, $\boldsymbol{\alpha} = \mathbf{w}$, which treats the mixture as a weighted average of the experts: a domain sampled half the time contributes half of its expert's change. Unlike the reward weights in reward composition settings, however, domain weights only set how often each domain is sampled during training, and training on a mixture is not a weighted average of training on each domain. Although all experts share the same initialization and training recipe, some move the policy many times as far as others (Figure 3). We therefore expect a domain whose expert moves the policy far on its own to take over more of the mixture than its domain weight suggests, and a domain whose expert barely moves the policy to take over less. We start from the domain weights and scale each one up or down by how far its expert moves. Figure 2 shows that the resulting coefficients fit the trained mixtures better than the domain weights themselves.

To measure how far expert k moves the policy, which we refer to as its *policy drift*, we use its KL divergence from the reference policy, summed over the set $\mathcal{H}$ of all prefixes in a fixed set of base model rollouts:

$$\mathrm{KL}_k \;=\; \sum_{h \in \mathcal{H}} \mathrm{KL}\big(\pi_{\mathrm{ref}}(\cdot \mid h) \,\big\|\, \pi_k(\cdot \mid h)\big) \;=\; -\sum_{h \in \mathcal{H}} \mathbb{E}_{v \sim \pi_{\mathrm{ref}}(\cdot \mid h)}\big[\phi_k(h, v)\big], \tag{4}$$

computed once per expert, exactly over the full vocabulary, with ϕ_k the policy log-ratio of Eq. (1). A composition that uses expert k alone with coefficient one is exactly π_k (Eq. (3)), so KL_k is how far that composition moves from π_{ref}. With coefficient α_k, it moves by about $\alpha_k^2 \, \mathrm{KL}_k$ when the drift is small. Because the KL averages over tokens drawn from π_{ref}, changes to tokens that π_{ref} rarely generates have little effect on the drift (Appendix B.2).

Concretely, we compare each expert's drift with the typical drift in the mixture and set $\boldsymbol{\alpha} = \tilde{\mathbf{w}}$, where

$$\tilde{w}_k \;=\; w_k \, \frac{\mathrm{KL}_k}{\prod_j \mathrm{KL}_j^{\,w_j}}, \tag{5}$$

The typical drift in the denominator is the $\mathbf{w}$-weighted geometric mean of the experts' drifts. An expert that moves the policy twice as far as the typical expert thus gets twice its domain weight, and one that moves half as far gets half. If all the drifts are the same, $\tilde{w}_k$ reduces to w_k. In an even Algebra-Graphs mixture, for example, the Algebra expert moves the policy further than the Graphs expert, so Algebra's coefficient rises above its domain weight and Graphs' falls below it.

Unless all the drifts are the same, the coefficients sum to more than one, by the weighted arithmetic–geometric mean inequality, so the composition puts a negative weight on π_{ref}. We predict $\widehat{\pi}_{\mathbf{w}} = \pi_{\mathrm{PM}(\tilde{\mathbf{w}})}$ (Eq. (3)), which needs only the reference model, the experts and $\mathbf{w}$: no expert parameters and no trained mixture. In practice, $\tilde{\mathbf{w}}$ comes close to coefficients fitted to the trained mixtures (Figures 2 and 4).

2.4 On-Policy Distillation

Beyond its use for mixture evaluation and selection, Policy-Merge can serve as a teacher for on-policy distillation (OPD). For distillation, we set the teacher weights to the data proportions that the mixture model is trained on, $\boldsymbol{\lambda} = \mathbf{w}$, rather than to the coefficients $\tilde{\mathbf{w}}$ of Section 2.3, so that the weights lie on the simplex. Weighted multi-teacher reverse KL distillation with weights $\mathbf{w}$ then has the same per-token student gradient as distillation from the single *composed teacher* $q_{\mathbf{w}}$. Let $h_t = (x, y_{<t})$ denote a prefix generated by the student policy π_θ, and let $\{\pi_k\}_{k=1}^K$ be teacher policies with full support over a shared vocabulary $\mathcal{V}$. For fixed weights $\boldsymbol{\lambda} \in \Delta^{K-1}$, the reference term in Eq. (3) vanishes, and the composed teacher is

$$q_{\boldsymbol{\lambda}}(v \mid h_t) = \pi_{\mathrm{PM}(\boldsymbol{\lambda})}(v \mid h_t) = \frac{\prod_{k=1}^K \pi_k(v \mid h_t)^{\lambda_k}}{Z_{\boldsymbol{\lambda}}(h_t)}, \tag{6}$$

where $Z_{\boldsymbol{\lambda}}(h_t)$ normalizes the distribution over $\mathcal{V}$. Consider a weighted reverse KL loss in which every teacher is evaluated at the same fixed prefix h_t, which we omit from the notation. Since the weights sum to one and $\log q_{\boldsymbol{\lambda}}(v) = \sum_k \lambda_k \log \pi_k(v) - \log Z_{\boldsymbol{\lambda}}$ (Eq. (6)),

$$\sum_{k=1}^K \lambda_k \,\mathrm{KL}\big(\pi_\theta \,\|\, \pi_k\big) = \mathbb{E}_{v\sim\pi_\theta}\Big[\log \pi_\theta(v) - \sum_{k=1}^K \lambda_k \log \pi_k(v)\Big] = \mathrm{KL}\big(\pi_\theta \,\|\, q_{\boldsymbol{\lambda}}\big) - \log Z_{\boldsymbol{\lambda}}. \tag{7}$$

For fixed teachers, weights, and prefix h_t, $Z_{\boldsymbol{\lambda}}(h_t)$ is independent of θ. Thus, reverse KL distillation from Policy-Merge has exactly the same student gradient as the corresponding weighted sum of teacher-specific reverse KL losses at that prefix (Appendix F.1).

We then distill the composed teacher $q_{\mathbf{w}}$ into a single student through on-policy reverse KL updates (Figure 11). The experts are still needed during distillation, to compute $q_{\mathbf{w}}$ on the prefixes the student generates, but the distilled student runs on its own at deployment. Unlike domain-routed multi-teacher OPD, where a specialist teacher is selected for each prompt, the composed teacher combines contributions from all domain experts at every token.

3 Experiments

3.1 Setup

We post-train `Qwen3-0.6B` (Yang et al., 2025) on six Reasoning Gym (Stojanovski et al., 2026) domains (*algebra*, *geometry*, *induction*, *cognition*, *arithmetic* and *graphs*), each with two or three tasks and 10,000 training prompts. We train one expert per domain and, as ground truth, eight mixtures of two to six domains, all by GRPO without a KL penalty, with LoRA, for 156 steps (Figure 5, Appendix A). We also use Olmo 3's released RL-Zero 7B Math, Code and Instruction-Following experts (full-parameter RL) and RL-Zero Mix, trained from the same base on all three (Olmo et al., 2025), and the three SmolLM3 teachers and distilled student of Open-MOPD (Gao et al., 2026).

3.2 Experiments and Baselines

Similarity to trained mixtures. We compare Policy-Merge at $\boldsymbol{\alpha} = \tilde{\mathbf{w}}$ with the trained mixtures in accuracy on 200 problems per task and in per-token KL on their samples (Appendix A.2). Composition at $\boldsymbol{\alpha} = \mathbf{w}$ isolates the coefficients, and weight merging, $\theta_{\mathrm{ref}} + \sum_k w_k \Delta\theta_k$, isolates composing in policy rather than parameter space.

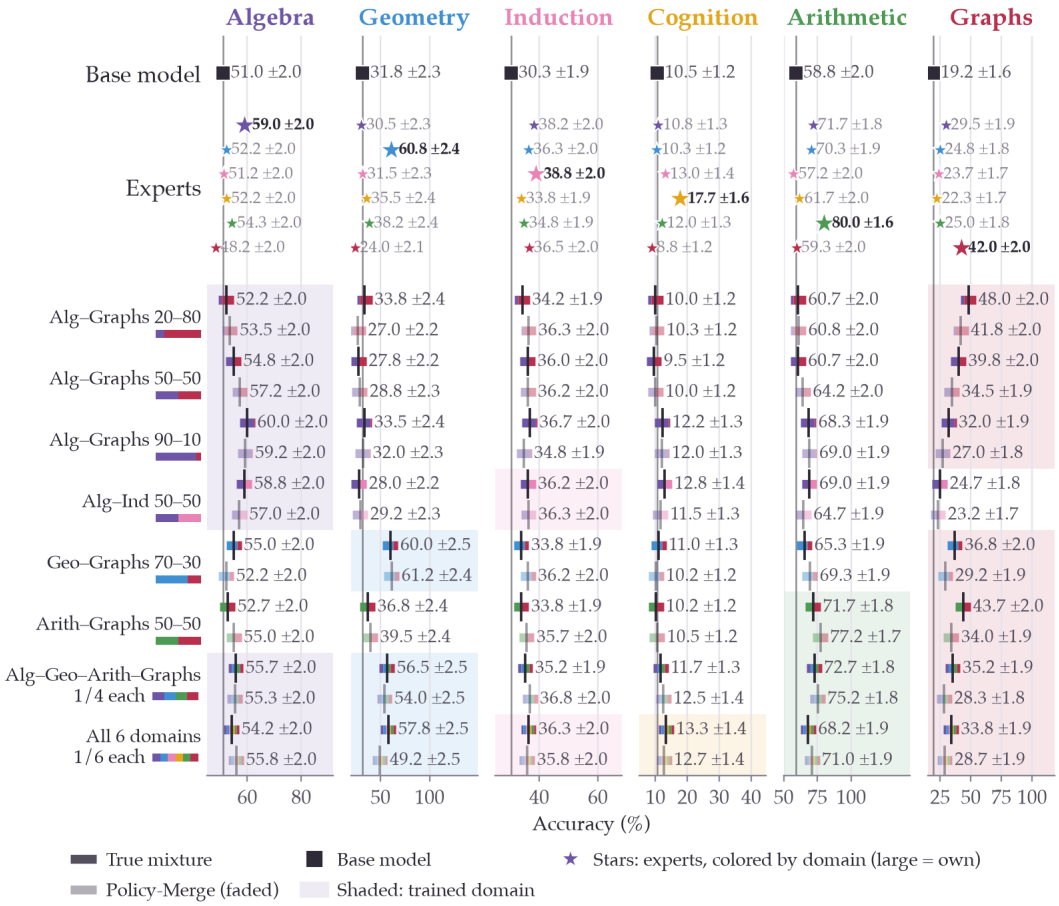


Figure 5: **POLICY-MERGE approximates policies trained on data mixtures**: Composing the experts results in accuracy close to the trained policy on both trained-on and out-of-mixture domains.

Mixture selection. For MATH-500, we decode 40 candidate compositions at their data proportions ($\boldsymbol{\alpha} = \mathbf{w}$) on 150 of its problems, fit a regressor from weights to accuracy, train the predicted best mixture for 156 steps, and evaluate it on the other 350 problems. We compare with MergeMix (Wang et al., 2026b) and OP-Mix (Hu et al., 2026), adapted to RL (Appendix E.2).

Distillation. We distill the composition at $\boldsymbol{\alpha} = \mathbf{w}$ (Section 2.4) into a LoRA student for 156 steps on Algebra/Graphs 20/80, 50/50 and 90/10, and compare it with decoding the composition and with routed-expert MOPD (Ma et al., 2026), which distills each prompt from its domain's expert.

External models. We compose the Olmo 3 (Olmo et al., 2025) experts with $\tilde{\mathbf{w}}$ (normalized to sum to one) and the Open-MOPD teachers with equal weights, and compare with RL-Zero Mix and the distilled student.

4 RESULTS

We first show that POLICY-MERGE reproduces the policy of training on a mixture, in accuracy and in distribution (Section 4.1). We then use it to select mixtures (Section 4.2) and as a teacher for on-policy distillation (Section 4.3), and show that it extends to larger models (Section 4.4).

4.1 POLICY-MERGE PREDICTS MULTI-DOMAIN RL OUTCOMES

Similarity in accuracy. If POLICY-MERGE predicts training, it should match the trained mixture's accuracy on every domain, including those outside the mixture. Averaged across the 17 tasks, the accuracy of POLICY-MERGE has a mean absolute error of 0.93 points relative to the eight trained mixtures (Figure 5, Table 3). Choosing to compose policies rather than weights and rescaling each domain weight by its expert's policy drift (Section 2.3) have both better approximated the true policy. With the domain weights as coefficients ($\boldsymbol{\alpha} = \mathbf{w}$), the error rises to 1.34 points, and with merged weights instead of composed policies to 2.13. Our policy approximation also holds as the number

Table 1: **POLICY-MERGE has the lowest KL to every trained mixture**: Policy distance to POLICY-MERGE is smaller than to weight merging or using domain weights ($\boldsymbol{\alpha} = \mathbf{w}$) for policy mixing

	Forward KL(Trained ‖ ·)			Reverse per-state KL(· ‖ Trained)		
	POLICY-MERGE ($\boldsymbol{\alpha} = \mathbf{w}$)	POLICY-MERGE ($\boldsymbol{\alpha} = \tilde{\mathbf{w}}$)	Weight merging (at $\mathbf{w}$)	POLICY-MERGE ($\boldsymbol{\alpha} = \mathbf{w}$)	POLICY-MERGE ($\boldsymbol{\alpha} = \tilde{\mathbf{w}}$)	Weight merging (at $\mathbf{w}$)
Reference: seed to seed	0.0058			0.0058		
Two-domain mixtures						
Alg 50 / Gra 50	0.0158 (5.0×)	0.0094 (8.4×)	0.0191 (4.1×)	0.0169 (6.9×)	0.0099 (11.9×)	0.0211 (5.6×)
Alg 90 / Gra 10	0.0044 (18.6×)	0.0041 (20.2×)	0.0051 (16.2×)	0.0046 (26.8×)	0.0042 (29.7×)	0.0053 (23.2×)
Alg 50 / Ind 50	0.0066 (12.1×)	0.0054 (14.7×)	0.0074 (10.9×)	0.0069 (17.4×)	0.0055 (21.5×)	0.0078 (15.4×)
Geo 70 / Gra 30	0.0101 (8.3×)	0.0054 (15.4×)	0.0127 (6.6×)	0.0106 (11.9×)	0.0055 (23.0×)	0.0136 (9.3×)
Multi-domain mixtures						
Four-way	0.0222 (4.0×)	0.0061 (14.6×)	0.0308 (2.8×)	0.0246 (5.3×)	0.0062 (21.2×)	0.0356 (3.6×)
Six-way	0.0272 (3.7×)	0.0099 (10.4×)	0.0335 (3.0×)	0.0312 (4.9×)	0.0105 (14.9×)	0.0396 (3.9×)
Geometric mean	0.0119 (7.2×)	0.0064 (13.4×)	0.0145 (5.9×)	0.0128 (10.0×)	0.0066 (19.5×)	0.0160 (8.0×)

of domains grows: for the uniform four- and six-domain mixtures, POLICY-MERGE is within 2.5 points of the trained mixture on five of the six domains and within 2.8 points on four of the six.

Distributional closeness. Similar accuracy could come from different policies, so we also compare the policies directly. As a reference, the KL between the base model and the trained policy measures how far training moved the policy. POLICY-MERGE has the lowest KL on every mixture, in both directions (Table 1). Forward KL penalizes missing tokens that the trained policy uses, and reverse KL penalizes mass on tokens it avoids. In forward KL, it is 13.4× closer to the trained policy than the base model is (geometric mean, 8.4–20.2× per mixture), so its KL is about 7.5% of the base model's. Composition at the domain weights is 7.2× closer, and weight merging 5.9×.

4.2 OPTIMIZING DATA MIXTURES

Selecting a mixture usually requires a full RL run for each candidate. Since POLICY-MERGE predicts the trained policy, it can score candidates simply by decoding their compositions at the data proportions ($\boldsymbol{\alpha} = \mathbf{w}$), so only the predicted best mixture is trained (Appendix E.1). For MATH-500, a benchmark outside our six training domains, scoring 40 candidates this way generates 850–1,070× fewer tokens than training all of them (Appendix E.3).

After training, the mixture selected by POLICY-MERGE, 59% Geometry and 41% Arithmetic, reaches 70.6 ± 0.8% on the 350 held-out MATH-500 problems, 4.6 points above the base model (Figure 6). MergeMix and OP-Mix select from the same experts, scoring candidates with merged weights and with short probe training runs (Appendix E.2). MergeMix's mixture gains slightly less (70.1±0.8%), within noise of POLICY-MERGE, and OP-Mix's less than half as much (68.0±1.5%). Scoring composed policies is thus enough to select a mixture that trains well.

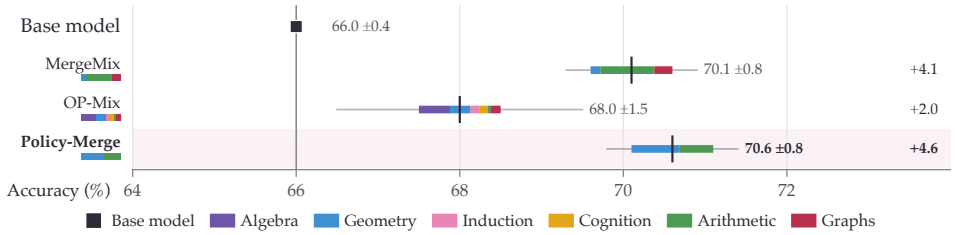


Figure 6: **POLICY-MERGE can be used to select a mixture for targeting MATH-500 performance**: Each method selects a mixture of the six domains through its proxies, and we train a model on each selection. The right column shows the gain over the base model. Error bars: SD over four greedy runs with different prompt orders.

4.3 POLICY-MERGE AS TEACHER FOR MULTI-TEACHER ON-POLICY DISTILLATION

Weighted multi-teacher OPD has the same per-token objective as distilling the single composed teacher POLICY-MERGE (Section 2.4), so POLICY-MERGE is the policy the student is trained toward. With the same number of steps as RL training, students distilled from POLICY-MERGE are 0.8, 0.8 and 2.1 points lower than decoding it at Algebra/Graphs 20/80, 50/50 and 90/10, averaged across tasks (Figure 7, Table 5). Routed-expert MOPD, which sends each prompt to the expert of its own domain, is within 0.7 points of the decoded composition. Decoding POLICY-MERGE thus shows what distillation will reach before any student is trained.

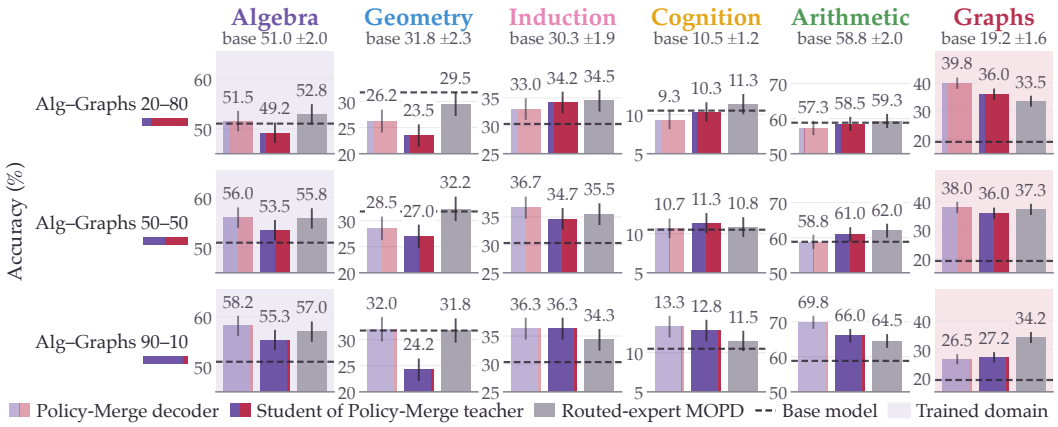


Figure 7: **Decoding POLICY-MERGE allows for predicting the distilled student's capabilities**: Averaged across tasks, POLICY-MERGE is about as accurate as routed-expert MOPD and slightly more accurate than the student distilled from it.

4.4 BEYOND OUR TRAINING SETUP

POLICY-MERGE extends to external 7B experts. Olmo 3 provides an independently trained ground-truth mixture, RL-Zero Mix, together with the experts for its domains. RL-Zero Mix is trained from the same base on a mixture of the data used to train three released experts (Math, Code, Instruction-Following). Composing these experts with no training, POLICY-MERGE reaches 54.5% on 200 MATH-500 problems and 43.1% on MBPP+, against 53.0% and 44.2% for RL-Zero Mix, within one standard error on both benchmarks (Figure 8). These experts are about 12× larger than ours and trained with full-parameter RL rather than LoRA, so POLICY-MERGE prediction capabilities are not tied to our setup.

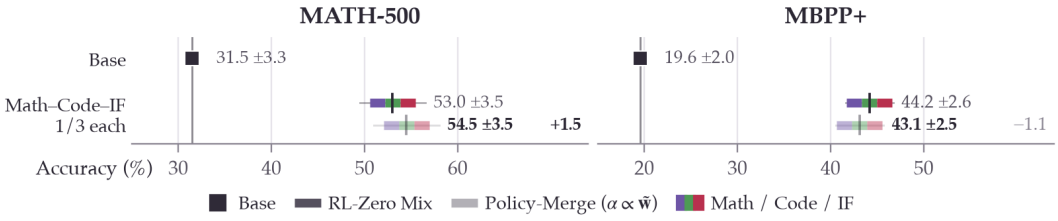


Figure 8: **At 7B, POLICY-MERGE matches training on the mixture**: Composing Olmo 3's released Math, Code and Instruction-Following experts, with no training, performs on par with RL-Zero Mix, which is trained on all three domains.

Composing released teachers matches a released distilled student. Multi-teacher distillation is costly, so it's helpful to predict in advance what capabilities the student will have. Our previous distillation results use students we trained ourselves. Open-MOPD (Gao et al., 2026) provides an independent check since it releases three SmolLM3 teachers and the student distilled from them, which is trained by routing each prompt to the teacher of its domain. Decoding the equal-weight composition of these teachers, with no training, scores 31.7% overall against the student's 31.5% under the same harness (Figure 9), and is within 3.3 points of it on every benchmark. As in Section 4.3, the composition predicts a routed student, here one trained by others.

5 PRIOR WORK

RL Post-training and on-policy Distillation. Reinforcement learning with verifiable rewards (RLVR) (Lambert et al., 2024) post-trains language models on tasks with checkable answers, typically with GRPO-style objectives (Shao et al., 2024; Guo et al., 2025; Team et al., 2025). On-policy distillation (OPD) trains a student on its own samples under a teacher's token-level reverse KL (Agarwal et al., 2024; Gu et al., 2024; Lu & Lab, 2025), extending knowledge distillation, and is used to build smaller models (Yang et al., 2025). Multi-teacher variants distill several domain RL experts into one model (Ma et al., 2026; Xiao et al., 2026; Xiaomi MiMo Team, 2026; Gao et al., 2026; Wu et al., 2026). We show that weighted multi-teacher OPD is distillation from a single composed teacher, which can be decoded and evaluated before any student is trained.

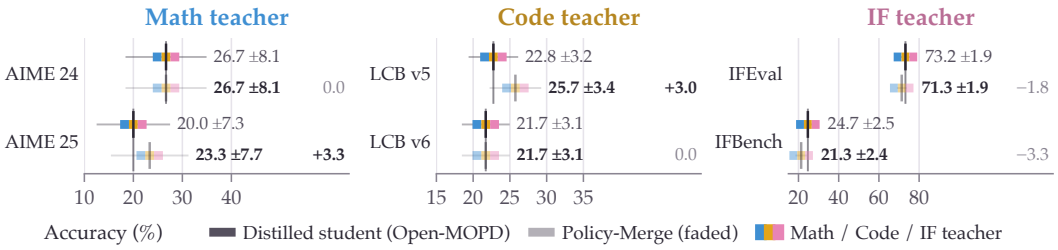


Figure 9: **Composing the teachers matches the distilled student**: Decoding an equal weight composition of OpenMOPD's three SmolLM3 teachers, with no training, performs on par with the student distilled from them.

Multi-domain post-training. Recent work compares RL on a mixture of domains with domain experts combined by model merging or multi-teacher distillation, and finds that the combined experts can match the accuracy of RL on the mixture (Wang et al., 2026a; Wu et al., 2026; Gu et al., 2026). Each of these studies uses one fixed mixture, while we compare mixtures without training on each one. A parameter merge (Ilharco et al., 2023) matches the weighted sum of the experts' logit changes only to first order. We instead compose these logit changes directly.

Decoding-time policy composition. Several methods combine models at decoding time. DExperts (Liu et al., 2021) forms a product of experts from a base model and attribute-specific experts, while proxy-tuning (Liu et al., 2024) transfers the logit shift of a small tuned model to a larger base model. Controlled decoding (Mudgal et al., 2024) combines models for multiple rewards at inference time, and multi-objective decoding (Shi et al., 2024) composes independently aligned policies for any weighting of their rewards. More closely related to our work, Hamidieh et al. (2026) compose policies trained on different rewards to predict the policy obtained by RL on a weighted combination of those rewards. We predict the policy obtained by RL on a data mixture instead, and rescale each domain weight by its expert's policy drift.

6 Discussion & Conclusion

Our results show that multi-domain RL post-training has a compositional structure in the policy space. This structure can be leveraged to predict outcomes of RL runs trained on a range of environments. Our results also suggest that part of what multi-teacher OPD learns can be represented directly by composing the teacher policies. This provides a training-free way to preview the behavior of a multi-teacher student, while distillation remains useful when a single model is needed.

Policy-Merge is the most useful when domain experts or different RL environments are given. The setting becomes even more useful for post-training pipelines in which environments are repeatedly added or updated. Composing several specialists increases inference cost during mixture evaluation, but this cost can be removed after selection by distilling the composition.

In short, we showed that the outcome of RL on a mixture of training environments can be predicted from policies trained separately on the individual environments. Policy-Merge does this by composing the expert policies directly at decoding time, and we find that the resulting policy closely approximates models trained on the corresponding mixtures in both downstream performance and distributional similarity. This makes it possible to evaluate and select candidate training mixtures, and provides a training-free approximation to multi-teacher on-policy distillation. Together, these results suggest that existing RL specialists can be reused to predict the policy of new post-training allocations before committing to a training run.

Limitations

Our controlled mixture experiments are primarily on 0.6B models and selected environments, although we also test one released 7B full parameter RLVR model trained on a mixture. Broader evaluation across model scales and more diverse RL environments would help establish how generally the result holds. We compare against eight trained mixtures with one training seed each, too few to tell apart how well Policy-Merge, composition at the data proportions and weight merging rank mixtures. Policy-Merge also assumes that the policies are trained from a shared reference model with the same recipe. Finally, we consider fixed environment mixtures throughout training, rather than curricula or adaptive mixtures that change over the course of RL.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods (coding agents were used), and plotting some figures given a concrete description. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to search the related literature, to check references and cross-references, and to give feedback on initial sections of our paper. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY

All models and data we use are public. Appendices A-F give the full training, decoding, selection and distillation details. We will release our code upon publication.

REFERENCES

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

Anyi Xu DeepSeek-AI, Bangcai Lin, Bing Xue, Bingxuan Wang, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *https://huggingface. co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4. pdf*, 2026.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, et al. Towards general agentic intelligence via environment scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, 2026.

Huan-ang Gao, Haohan Chi, Yong Yan, Shiyuan Feng, Hanlin Wu, Zheng Jiang, Bingxiang He, Wei-Ying Ma, Ya-Qin Zhang, and Hao Zhou. Open-mopd: Diagnosing and fixing capability imbalance in multi-teacher on-policy distillation. *arXiv preprint arXiv:2608.19098*, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Naibin Gu, Chenxu Yang, Qingyi Si, Chuanyu Qin, Dingyu Yao, Peng Fu, Zheng Lin, Weiping Wang, Nan Duan, and Jiaqi Wang. Co-evolving policy distillation. *arXiv preprint arXiv:2604.27083*, 2026.

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

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

Kimia Hamidieh, Giannis Daras, and Antonio Torralba. Poem: Predicting rl outcomes from existing policies, 2026. URL https://arxiv.org/abs/2609.30226.

Tom Heskes. Selecting weighting factors in logarithmic opinion pools. volume 10, 1997.

Michael Y. Hu, Apurva Gandhi, Kyunghyun Cho, Tal Linzen, and Pratyusha Sharma. Always learning, always mixing: Efficient and simple data mixing all the time. 2026. URL https://arxiv.org/abs/2605.15220.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. 2023. URL https://arxiv.org/abs/2212.04089.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Zhaoyi Li, Deyang Kong, Yuan Wei, Evan Yang, Ranran Shen, Mahardika Krisna Ihsani, Ming Yang, Wei Zhang, Chuan Hao, Jian Yang, et al. Every coin has two sides: On the dual nature of generalization in on-policy distillation of large language models. *arXiv preprint arXiv:2608.16647*, 2026.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. Dexperts: Decoding-time controlled text generation with experts and anti-experts. 2021. URL https://arxiv.org/abs/2105.03023.

Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A. Smith. Tuning language models by proxy. 2024. URL https://arxiv.org/abs/2401.08565.

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

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

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.

Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, Jilin Chen, Alex Beutel, and Ahmad Beirami. Controlled decoding from language models. 2024. URL https://arxiv.org/abs/2310.17022.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious. 2025. URL https://arxiv.org/abs/2501.00656.

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

Ruizhe Shi, Yifang Chen, Yushi Hu, Alisa Liu, Hannaneh Hajishirzi, Noah A Smith, and Simon S Du. Decoding-time language model alignment with multiple objectives. volume 37, pp. 48875–48920, 2024.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. volume 38, 2026.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Haoqing Wang, Xiang Long, Ziheng Li, Yilong Xu, Tingguang Li, and Yehui Tang. To mix or to merge: Toward multi-domain reinforcement learning for large language models. 2026a. URL https://arxiv.org/abs/2602.12566.

Jiapeng Wang, Changxin Tian, Kunlong Chen, Ziqi Liu, Jiaxin Mao, Wayne Xin Zhao, Zhiqiang Zhang, and Jun Zhou. Mergemix: Optimizing mid-training data mixtures via learnable model merging. 2026b. URL https://arxiv.org/abs/2601.17858.

Siye Wu, Kai Yang, Yuchen Cai, Xin Xu, Peng-Yuan Wang, Jiaxuan Wang, Jiashun Liu, Jiafei Lyu, Yangkun Chen, Saiyong Yang, et al. Consolidating rlvr capabilities across domains: A deep dive into fusion paradigms. *arXiv preprint arXiv:2608.27409*, 2026.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Xiaomi MiMo Team. Mimo-v2.6-pro-rl. https://huggingface.co/XiaomiMiMo/MiMo-V2.6-Pro-RL, 2026.

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

Ningyuan Yang, Weihua Du, Weiwei Sun, Sean Welleck, and Yiming Yang. Gradalign: Gradient-aligned data selection for llm reinforcement learning. *arXiv preprint arXiv:2602.21492*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. Rlve: Scaling up reinforcement learning for language models with adaptive verifiable environments. 2026. URL https://arxiv.org/abs/2511.07317.

A Experimental Details

A.1 Reasoning Gym Domains and Training

Domains. We use `Qwen3-0.6B` (Yang et al., 2025) as the base model and construct six post-training domains from Reasoning Gym (Stojanovski et al., 2026): *algebra*, *geometry*, *induction*, *cognition*, *arithmetic*, and *graphs*. Algebra contains simple equations, polynomial equations, and polynomial multiplication tasks. Geometry contains simple and advanced geometry tasks. Induction contains number sequences, modulo grids, and ACRE tasks. Cognition contains cube rotation, Figlet recognition, and rectangle counting tasks. Arithmetic contains chain sums, basic arithmetic, and decimal arithmetic tasks. Graphs contains family relationships, largest-island problems, and course scheduling.

Data and training. We construct a dataset of $10{,}000$ training prompts for each domain, distributing examples as evenly as possible among its component tasks. All experts are initialized from the same base model and are post-trained for their corresponding domain via GRPO without a KL penalty, with LoRA rank 64 and scaling parameter 128. Training uses 156 optimization steps, a prompt batch size of 64, 16 sampled responses per prompt, a learning rate of 10^{-6}, and a maximum response length of 4,096 tokens. Responses are sampled with temperature 1 and top-p equal to 1. We enable Qwen's reasoning mode. Trained-mixture models use the same recipe on the mixed data.

Reward and accuracy. The training reward is the Reasoning Gym task verifier's score in $[0, 1]$, used as is: 1 for a correct answer, with partial credit on some tasks, and no format term. Evaluation accuracy counts an answer as correct only when this score is 1.

A.2 Evaluation and KL

We evaluate the base model, individual experts, trained-mixture models, and Policy-Merge policies on the held-out validation sets of the Reasoning Gym domains using greedy decoding, thinking, and a maximum generation length of 4,096 tokens. The same validation sets are used to assess performance similarity for mixtures themselves.

To assess distributional similarity, we sample 200 trajectories in proportion to the mixture's domains from each trained-mixture model and evaluate every composed policy on the same resulting prefixes. Our primary distributional metric is forward conditional KL on prefixes sampled from the trained mixture, Eq. (8). At each token of the trajectory, we compute the KL divergence using the complete vocabulary distributions:

$$D_{\rightarrow} = \mathbb{E}_{x,y\sim\pi_{\mathbf{w}}}\left[\frac{1}{|y|}\sum_{t}\mathrm{KL}\big(\pi_{\mathbf{w}}(\cdot \mid h_t)\,\big\|\,\widehat{\pi}_{\mathbf{w}}(\cdot \mid h_t)\big)\right], \tag{8}$$

where $h_t = (x, y_{<t})$ denotes the prefix at step t. The reverse column of Table 1 is the per-state reverse KL, $\mathrm{KL}(\widehat{\pi}_{\mathbf{w}}(\cdot \mid h_t) \,\|\, \pi_{\mathbf{w}}(\cdot \mid h_t))$, evaluated on the same trained-mixture prefixes. It is not a sequence-level reverse KL under the composed policy's own samples.

Policy drift. We measure each expert's policy drift (Eq. (4)) on the prefixes of 200 rollouts sampled from the base model.

A.3 Olmo 3 RL-Zero

To test whether our findings extend beyond the 0.6B Reasoning Gym setting, we additionally evaluate released Olmo 3 RL-Zero 7B policies. This increases model scale by more than an order of magnitude and replaces short LoRA adaptation on synthetic tasks with independently trained, full-parameter RLVR specialists. All released policies originate from the same Olmo 3 Base 7B checkpoint. We use the released Math, Code, and instruction-following specialists and compare their equal-weight policy-space composition against Ai2's independently trained Olmo 3 RL-Zero Mix checkpoint and the shared Base model.

B Composition and Coefficients

B.1 Why Compose Policies Rather Than Merge Parameters

Each expert $\theta_k = \theta_{\text{ref}} + \Delta\theta_k$ is post-trained from the reference parameters θ_{ref}, whose policy is π_{ref}, with policy $\pi_k(\cdot \mid h_t) = \text{softmax}(z_k(h_t))$ at prefix $h_t = (x, y_{<t})$ and policy log-ratio $\phi_k = \log \pi_k - \log \pi_{\text{ref}}$ (Eq. (1)). The composition of Eq. (3), with a temperature τ on the composed shift, is

$$q_{\boldsymbol{\alpha},\tau}(\cdot \mid h_t) = \text{softmax}\big(\log \pi_{\text{ref}} + \tfrac{1}{\tau}\textstyle\sum_k \alpha_k \phi_k\big) = \text{softmax}\big(z_{\text{ref}} + \tfrac{1}{\tau}\textstyle\sum_k \alpha_k (z_k - z_{\text{ref}})\big), \qquad (9)$$

since ϕ_k and $z_k - z_{\text{ref}}$ differ by a constant the softmax cancels, and $q_{\boldsymbol{\alpha},1} = \pi_{\text{PM}(\boldsymbol{\alpha})}$. The temperature τ scales the composed shift, not z_{ref}. All our results use $\tau = 1$. Weight merging instead uses $\bar{\theta} = \theta_{\text{ref}} + \sum_k \alpha_k \Delta\theta_k$, so $\pi_{\bar{\theta}} = \text{softmax}(\bar{z} - \Gamma_{\boldsymbol{\alpha}})$ with $\bar{z} = z_{\text{ref}} + \sum_k \alpha_k (z_k - z_{\text{ref}})$ and $\Gamma_{\boldsymbol{\alpha}} = \bar{z} - z(\bar{\theta}; h_t)$. The two agree only if $\Gamma_{\boldsymbol{\alpha}}$ is constant across the vocabulary. Composition combines the experts' changes to the predictive distribution directly and needs no assumption that logits are linear in the weights.

B.2 Policy Drift and the Composition

Section 2.3 scales each domain weight by the policy drift KL_k of the domain's expert (Eq. (4)). This appendix explains why KL_k is the right measure for this: it describes how far the composition itself moves away from π_{ref} when expert k is added. Everything below happens at a single prefix h, which we drop from the notation. KL_k adds up the result over the prefixes in $\mathcal{H}$. At one prefix, the composition of Eq. (3) is

$$\pi_{\boldsymbol{\alpha}}(v) = \frac{\pi_{\text{ref}}(v)\exp\big(\sum_k \alpha_k \phi_k(v)\big)}{Z_{\boldsymbol{\alpha}}}, \qquad Z_{\boldsymbol{\alpha}} = \mathbb{E}_{v\sim\pi_{\text{ref}}}\Big[\exp\Big(\sum_k \alpha_k \phi_k(v)\Big)\Big].$$

An expert on its own. Start with the simplest composition: coefficient one on expert k and zero on all others, which we write $\pi_{\mathbf{e}_k}$. Its log-ratio is $\phi_k = \log \pi_k - \log \pi_{\text{ref}}$, so adding it to $\log \pi_{\text{ref}}$ gives back $\log \pi_k$, and the composition is simply expert k:

$$\pi_{\mathbf{e}_k}(v) = \frac{\pi_{\text{ref}}(v)\,\pi_k(v)/\pi_{\text{ref}}(v)}{\sum_{v'} \pi_k(v')} = \pi_k(v), \qquad \text{so} \qquad \text{KL}\big(\pi_{\text{ref}} \,\|\, \pi_{\mathbf{e}_k}\big) = \text{KL}\big(\pi_{\text{ref}} \,\|\, \pi_k\big).$$

Added up over the prefixes, the right-hand side is KL_k. So the policy drift is exactly how far the composition moves from π_{ref} when it uses expert k at full strength.

Scaling the coefficient. In practice the coefficients are not one. If expert k enters with coefficient a, the composition moves by

$$\text{KL}\big(\pi_{\text{ref}} \,\|\, \pi_{a\mathbf{e}_k}\big) = \log \mathbb{E}_{\pi_{\text{ref}}}\big[e^{a\phi_k}\big] - a\,\mathbb{E}_{\pi_{\text{ref}}}[\phi_k] \;\approx\; \frac{a^2}{2}\,\text{Var}_{\pi_{\text{ref}}}(\phi_k),$$

where the first step writes out the KL using the definition of $\pi_{a\mathbf{e}_k}$, and the second keeps the leading term of $\log \mathbb{E}[e^{aX}] = a\,\mathbb{E}[X] + \frac{a^2}{2}\,\text{Var}[X] + O(a^3)$. Setting $a = 1$ gives $\text{KL}(\pi_{\text{ref}} \,\|\, \pi_k) \approx \frac{1}{2}\,\text{Var}_{\pi_{\text{ref}}}(\phi_k)$, so

$$\text{KL}\big(\pi_{\text{ref}} \,\|\, \pi_{a\mathbf{e}_k}\big) \;\approx\; a^2\,\text{KL}\big(\pi_{\text{ref}} \,\|\, \pi_k\big).$$

Halving an expert's coefficient therefore moves the composition about a quarter as far. The drift at full strength sets the scale of the expert's effect at every coefficient.

What the variance measures. The variance above is taken over the next token v, drawn from π_{ref} at the fixed prefix h:

$$\text{Var}_{\pi_{\text{ref}}}(\phi_k) = \sum_v \pi_{\text{ref}}(v)\big(\phi_k(v) - \mathbb{E}_{\pi_{\text{ref}}}[\phi_k]\big)^2.$$

It is large when the expert treats the tokens the reference model would likely generate very differently, making some much more likely and others much less likely. For example, if π_{ref} gives two tokens probability 0.5 each and the expert changes them to 0.9 and 0.1, their log-ratios are far apart and the variance is large. It is small when the expert changes all likely tokens by about the same amount, which normalization undoes anyway. Because each token is weighted by $\pi_{\text{ref}}(v)$, a token

that the reference model almost never generates barely counts, however much the expert changes it. This matches the composition, which samples from π_{ref} reweighted by $\exp(\sum_k \alpha_k \phi_k)$: such a token remains almost never sampled. Eq. (4) computes the drift exactly as $-\mathbb{E}_{\pi_{\text{ref}}}[\phi_k]$, how much less likely the expert finds the reference model's own choices. The variance form is its approximation for small drift, and is easier to interpret.

Several experts. With several experts, the same expansion gives

$$\mathrm{KL}\big(\pi_{\text{ref}} \,\|\, \pi_{\boldsymbol{\alpha}}\big) \;\approx\; \frac{1}{2}\,\mathrm{Var}_{\pi_{\text{ref}}}\Big(\sum_k \alpha_k \phi_k\Big) \;=\; \frac{1}{2}\sum_{j,k} \alpha_j \alpha_k\, \mathrm{Cov}_{\pi_{\text{ref}}}(\phi_j, \phi_k).$$

The terms with $j \neq k$ describe how the experts' changes line up with each other. KL_k does not capture them: it measures each expert on its own.

Summary. The policy drift measures how strongly an expert pulls the composition away from π_{ref}. It needs only π_{ref} and π_k, and its weighting by π_{ref} comes from the form of the composition rather than being a separate choice. The coefficients $\tilde{\mathbf{w}}$ use it to decide how much each expert counts: a domain whose expert moves the policy far on its own also moves the trained mixture more than its domain weight suggests, so $\boldsymbol{\alpha} = \mathbf{w}$ underweights it and $\tilde{\mathbf{w}}$ gives it a larger coefficient.

B.3 Details for Figures 2–4

All three figures use the prefixes $\mathcal{H}$ of the 200 rollouts sampled from π_{ref} described under *Policy drift* above: one rollout per prompt, for 200 held-out validation prompts split evenly over the six domains (and evenly over each domain's tasks), sampled at temperature 1 and top-p 1 with thinking enabled and at most 4,096 new tokens. Every other policy, experts and trained mixtures alike, is teacher-forced on these rollouts, so no quantity below uses a trained mixture's own samples. All quantities use the complete next-token distributions over the full vocabulary at every response position and are summed over positions, and $|\mathcal{H}|$ is the total number of response tokens.

Policy drift (Figure 3). KL_k is Eq. (4), computed exactly: the full-vocabulary $\mathrm{KL}(\pi_{\text{ref}}(\cdot \mid h) \,\|\, \pi_k(\cdot \mid h))$ summed over all $h \in \mathcal{H}$. The figure shows $\mathrm{KL}_k / \max_j \mathrm{KL}_j$. The same values define $\tilde{\mathbf{w}}$ (Eq. (5)) in Figures 2 and 4.

Shift covariance. For a trained mixture $\pi_{\mathbf{w}}$, let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_K, \phi_{\mathbf{w}})$ and $G = \sum_{h \in \mathcal{H}} \mathrm{Cov}_{v \sim \pi_{\mathbf{w}}(\cdot|h)}\big[\boldsymbol{\phi}(h, v)\big]$, a $(K{+}1) \times (K{+}1)$ matrix in which each token is weighted by the *trained mixture's* probability. We write $G_{\phi\phi}$ for the expert block, $\mathbf{g}$ for the expert–mixture column and $G_{\mathbf{ww}}$ for the mixture entry. With the residual $e_{\boldsymbol{\alpha}} = \phi_{\mathbf{w}} - \sum_k \alpha_k \phi_k$, the identity $\mathrm{KL}(\pi_{\mathbf{w}} \,\|\, \pi_{\mathrm{PM}(\boldsymbol{\alpha})}) = \mathbb{E}_{\pi_{\mathbf{w}}}[e_{\boldsymbol{\alpha}}] + \log \mathbb{E}_{\pi_{\mathbf{w}}}[e^{-e_{\boldsymbol{\alpha}}}]$ at each prefix gives, to second order in $e_{\boldsymbol{\alpha}}$,

$$\mathrm{KL}_2(\boldsymbol{\alpha}) = \frac{1}{2|\mathcal{H}|}\sum_{h \in \mathcal{H}} \mathrm{Var}_{\pi_{\mathbf{w}}(\cdot|h)}\big[e_{\boldsymbol{\alpha}}(h, \cdot)\big] = \frac{G_{\mathbf{ww}} - 2\boldsymbol{\alpha}^\top \mathbf{g} + \boldsymbol{\alpha}^\top G_{\phi\phi} \boldsymbol{\alpha}}{2|\mathcal{H}|} \quad \text{nats per token.} \tag{10}$$

Because the covariance is centered at each prefix, the prefix-dependent $\log Z_{\boldsymbol{\alpha}}(h)$ of Eq. (2) drops out, and no intercept is fitted.

KL-weighted R^2 (Figure 2). $R^2(\boldsymbol{\alpha}) = 1 - \mathrm{KL}_2(\boldsymbol{\alpha})/\mathrm{KL}_2(\mathbf{0}) = 1 - (G_{\mathbf{ww}} - 2\boldsymbol{\alpha}^\top \mathbf{g} + \boldsymbol{\alpha}^\top G_{\phi\phi} \boldsymbol{\alpha})/G_{\mathbf{ww}}$, the fraction of the mixture's second-order divergence from π_{ref} that the composition accounts for. It is evaluated at $\boldsymbol{\alpha} = \mathbf{w}$, at $\boldsymbol{\alpha} = \tilde{\mathbf{w}}$, and at $\hat{\boldsymbol{\alpha}}$, which minimizes Eq. (10) by bounded least squares over the experts with $w_k > 0$, with each $\hat{\alpha}_k \in [0, 1]$, the sum unconstrained, and $\hat{\alpha}_k = 0$ for the other experts. This set contains $\mathbf{w}$ and, for all six mixtures, $\tilde{\mathbf{w}}$, so $R^2(\hat{\boldsymbol{\alpha}})$ upper-bounds the other two.

Rule strength (Figure 4). Along $\boldsymbol{\alpha} = \gamma \tilde{\mathbf{w}}$, Eq. (10) is the quadratic $\mathrm{KL}_2(\gamma \tilde{\mathbf{w}}) = A - B\gamma + C\gamma^2$ with $A = G_{\mathbf{ww}}/2|\mathcal{H}|$, $B = \tilde{\mathbf{w}}^\top \mathbf{g}/|\mathcal{H}|$ and $C = \tilde{\mathbf{w}}^\top G_{\phi\phi} \tilde{\mathbf{w}}/2|\mathcal{H}|$. Curves show it for $\gamma \in [0.8, 1.5]$. Dots mark the rule ($\gamma = 1$) and rings the closed-form minimizer $\gamma^\star = B/2C$, which requires the trained mixture. Figures 2 and 4 cover the same six mixtures: Alg 90 / Gra 10, Alg 50 / Ind 50, Geo 70 / Gra 30, Alg 50 / Gra 50, the equal-weight four-way mixture of Algebra, Geometry, Arithmetic and Graphs, and the equal-weight six-way mixture.

C Decoding and Implementation

C.1 Decoding Algorithm

For each token, the reference model and every expert in the mixture read the same prefix of tokens generated so far by the composed policy with its own KV cache and return full-vocabulary logits. The experts are evaluated in the same batch. We combine raw logits over the vocabulary, $\sum_j c_j z_j$ equals $\log \pi_{\mathrm{PM}(\boldsymbol{\alpha})}$ up to a constant. All compositions use their coefficients as given, with no temperature on the composed shift ($\tau = 1$ in Eq. (9)). For the Olmo experts, the coefficients $\tilde{\mathbf{w}}$ are normalized to sum to one (Section 4.4). The sampling temperature T is separate and is applied after composition. Reasoning Gym and MATH-500 evaluations decode greedily ($T = 0$) with thinking enabled. Sampled decoding uses no top-k or top-p truncation.

Algorithm 1 Autoregressive decoding with Policy-Merge

Require: Prompt x; reference policy π_{ref} and experts $\{\pi_k\}_{k=1}^{K}$; coefficients $\boldsymbol{\alpha} \in \mathbb{R}^K$; sampling temperature $T \geq 0$; maximum generation length L

$c_{\mathrm{ref}} \leftarrow 1 - \sum_{k=1}^{K} \alpha_k$; $\quad c_k \leftarrow \alpha_k$ for $k = 1, \ldots, K$ ▷ Eq. (3)
$\mathcal{A} \leftarrow \{j \in \{\mathrm{ref}, 1, \ldots, K\} : c_j \neq 0\}$ ▷ models with $c_j = 0$ are not run
Prefill x into a separate KV cache for each $j \in \mathcal{A}$; $\quad y \leftarrow \varnothing$
for $t = 1, \ldots, L$ **do**
  $h_t \leftarrow (x, y_{<t})$
  Obtain full-vocabulary logits $z_j(h_t)$ for all $j \in \mathcal{A}$ in one batch
  $\ell_t \leftarrow \sum_{j \in \mathcal{A}} c_j \, z_j(h_t)$ ▷ $\mathrm{softmax}(\ell_t) = \pi_{\mathrm{PM}(\boldsymbol{\alpha})}(\cdot \mid h_t)$
  if $T = 0$ **then** $y_t \leftarrow \arg\max_v \ell_t[v]$ **else** $y_t \sim \mathrm{softmax}(\ell_t / T)$
  $y \leftarrow (y_{<t}, y_t)$; append y_t to the KV cache of every $j \in \mathcal{A}$
  if y_t is an end-of-sequence token **then break**
end for
return generated response y

Custom vLLM decoder. vLLM does not natively support combining the logits of multiple LoRA experts for next token prediction. Thus, to support Policy-Merge we have built a custom vLLM V1 decoding pipeline. It does one request for each expert for each prompt and uses an in-engine logits processor to form the weighted full-vocabulary logits before sampling. The processor writes the combined logits back to each expert in the group, so that they all sample the same token and retain the same prefix. We share our vLLM pipeline in our supplementary materials.

C.2 Decoding Cost

To generate from Policy-Merge, every expert reads the same string of generated tokens thus far and we calculate its policy distribution over the next token. Then, we compose all these policies into one distribution, sample a single token, and append that token to every expert's response. Generating with K experts requires decoding from K models at once.

With our custom vLLM decoder, decoding each additional expert adds about $0.29\times$ the single-model time, so six experts take $2.45\times$ as long as one. The slowdown is due to each expert keeping the KV cache, so with K experts the GPU can work on only about $1/K$ as many prompts at once. By contrast, with LoRA experts sharing the same base model, parameter storage is approximately one base model plus K small adapters.

Table 2: Generation speed with K experts on one H200 GPU, using the simpler implementation (BBH and GSM8K, 400 prompts, responses of up to 12,288 tokens; median over runs). Prompts in flight is the number of prompts the GPU works on at the same time.

K	1	2	3	4	5	6
generated tokens/s	2,074	1,576	1,310	1,153	992	846
time relative to $K = 1$	1.00×	1.32×	1.58×	1.80×	2.09×	2.45×
prompts in flight	78	39	26	19	15	13

This cost is paid only while evaluating candidate compositions. After selection, distilling the composed policy into a single student (Section 2.4) removes it at deployment, at a cost at our 156-step

budget of 0.8–2.1 points (1.2 on average) of accuracy averaged across the 17 tasks on the three Algebra/Graphs mixtures, comparing the student of the composed teacher with decoding the $\boldsymbol{\alpha} = \mathbf{w}$ composition. Decoding the composition directly keeps the K-fold forward-pass and KV-cache cost.

D Similarity to Trained Mixtures

D.1 Composition at the Data Proportions

Figure 10 reports the per-domain results for the simplest choice of coefficients, $\boldsymbol{\alpha} = \mathbf{w}$, which uses the data proportions directly. The main-text results use the Policy-Merge rule (Section 2.3), this figure shows how well using just the data proportions approximate the true mixture.

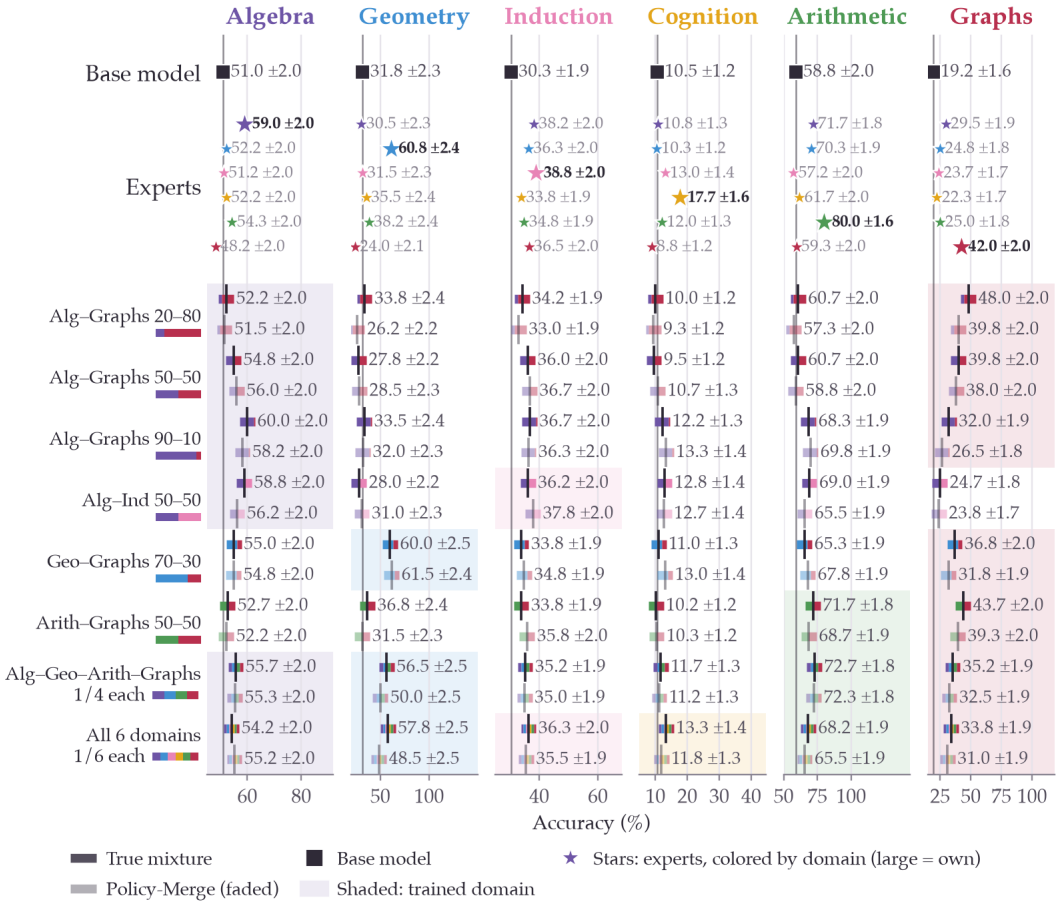


Figure 10: **Reasoning Gym validation of experts and mixtures, with Policy-Merge at $\boldsymbol{\alpha} = \mathbf{w}$.**

Table 3: **Ablation against weight merging.** Accuracy (%) averaged across the 17 Reasoning Gym tasks (3,400 items per mixture). Policy-Merge ($\boldsymbol{\alpha} = \tilde{\mathbf{w}}$) uses the coefficients of Eq. (5); Policy-Merge ($\boldsymbol{\alpha} = \mathbf{w}$) and weight merging (at $\mathbf{w}$), $\theta_{\text{ref}} + \sum_k w_k \Delta\theta_k$, both use the data proportions.

		Composed from experts		
	Trained mixture	Policy-Merge ($\boldsymbol{\alpha} = \tilde{\mathbf{w}}$)	Policy-Merge ($\boldsymbol{\alpha} = \mathbf{w}$)	Weight merging (at $\mathbf{w}$)
Per mixture (averaged across 17 tasks)				
Mean over 8 mixtures	41.24 ± 0.30	$\mathbf{40.48 \pm 0.30}$ $(\mathbf{-0.76})$	39.96 ± 0.30 (-1.28)	39.10 ± 0.30 (-2.13)
Mean |gap|	–	**0.93**	1.34	2.13
Spearman ρ vs trained	–	**0.83**	**0.83**	0.69
Kendall τ vs trained	–	**0.64**	**0.64**	0.50

D.2 KL ON THE COMPOSITION'S OWN SAMPLES

As a complement to Eq. (8), we also sample 200 trajectories from POLICY-MERGE itself and compute the same per-token forward KL on those prefixes. On the two-domain mixtures the KL is within 0.0003 nats/token of the values in Table 1 (Table 4).

Table 4: **The KL barely changes when POLICY-MERGE generates the prefixes**: Forward KL (nats per token) between the trained mixture and POLICY-MERGE, on prefixes sampled from the trained mixture (as in Table 1) and from POLICY-MERGE itself.

Mixture	Trained-mixture prefixes	POLICY-MERGE prefixes
Alg 90 / Gra 10	0.0041	0.0041
Alg 50 / Ind 50	0.0054	0.0055
Geo 70 / Gra 30	0.0054	0.0054
Alg 50 / Gra 50	0.0094	0.0097

E MIXTURE SELECTION

E.1 BENCHMARK-GUIDED OPTIMAL MIXTURE SELECTION

Let $\{\pi_k\}_{k=1}^K$ denote the policies for each of the K domain post-trained experts. We consider candidate mixtures $\boldsymbol{\alpha} \in \Delta^{K-1}$, where $\alpha_k \geq 0$ and $\sum_k \alpha_k = 1$. For selection, POLICY-MERGE scores each candidate at its data proportions, composing these experts at inference time via a token-level product of expert policies, $\pi_{\boldsymbol{\alpha}}(y_t \mid h_t) \propto \prod_{k=1}^K \pi_k(y_t \mid h_t)^{\alpha_k}$, which is Eq. (3) with coefficients that sum to one, to approximate the trained mixture.

To identify the optimal mixture for a target benchmark $\mathcal{B}$, we use a regression-based optimization. We first construct a deterministic set of $M = 40$ mixtures, $\mathcal{A}_{\text{seed}} = \{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(M)}\}$. This set includes the simplex vertices (100% weight on a single domain), the uniform mixture, pairwise $50/50$ mixtures, and an evenly spaced subset of mixtures at 10% increments. We also use this exact same candidate set for our MergeMix baseline (Appendix E.2).

Each candidate POLICY-MERGE policy is evaluated on a frozen benchmark-selection split, $\mathcal{D}_{\mathcal{B}}^{\text{sel}}$. For MATH-500, this split has 150 problems, and the other 350 are not used for selection. Using these M mixture observations, we fit a LightGBM regressor, $\widehat{f}_{\mathcal{B}} : \Delta^{K-1} \to [0, 1]$. We then define our selected POLICY-MERGE mixture, $\boldsymbol{\alpha}^\star$, as the predicted optimum over a fine 0.01 simplex lattice (a grid of all possible mixture combinations at exactly 1% increments):

$$\boldsymbol{\alpha}^\star = \arg\max_{\boldsymbol{\alpha} \in \Delta_{0.01}^{K-1}} \widehat{f}_{\mathcal{B}}(\boldsymbol{\alpha}) \tag{11}$$

The selected weights are then used directly as the data proportions of the trained mixture, without mapping them through the rule of Eq. (5).

E.2 ADAPTING MERGEMIX AND OP-MIX TO RL POST-TRAINING

MergeMix (Wang et al., 2026b) and OP-Mix (Hu et al., 2026) select pre-training and mid-training mixtures by approximating a mixture-trained model with a weighted parameter merge, $\theta_{\text{ref}} + \sum_k \alpha_k \Delta\theta_k$, and optimizing a fitted predictor of its performance. We re-implemented both for RL post-training, using the same six GRPO LoRA experts, selection splits and scorer as POLICY-MERGE. Each method's selected weights are used as data ratios for a 10k-prompt mixture, which is trained for the same 156 updates.

MergeMix. We merge the step-156 LoRA experts exactly, by concatenating their low-rank factors. We then score the same 40 candidate mixtures on the same 150 MATH-500 problems, with the same LightGBM fit and 1% simplex search as POLICY-MERGE (Appendix E.1). The only difference is how a candidate is scored: with the weight-merged model instead of by decoding the composition of the experts (Algorithm 1). Selection thus isolates composing policies from merging parameters.

OP-Mix. We train a 31-update prefix on uniform data ($0.2\times$ the budget). On top of it we train seven 16-update GRPO probes: one on the current mixture, and one per domain on $0.9\times$ that domain $+0.1\times$ the current mixture. We then score 20 Dirichlet-sampled interpolations of these probes and apply the upstream log-linear fit and KL-regularized proposal ($\lambda = 0.05$). We make two changes: the loss is $1-$accuracy rather than validation loss, and the source-expansion map allows overlapping probe mixtures.

E.3 Compute Cost of Mixture Search

Training one mixture takes 27.7 H200 GPU-hours on average (26.4–29.1), so training all 40 candidates would take about 1,109 GPU-hours, excluding queueing and failed runs. Policy-Merge instead decodes the 40 compositions on 150 selection problems: 6,000 generations and at most 24.6M tokens (30.7M with the forced-answer pass), with no gradient updates. Each trained mixture generates $156 \times 64 \times 16 = 159{,}744$ rollouts, up to 654M tokens, so the 40 candidates would need up to 26.2B tokens, 850–1,070$\times$ more. This assumes the experts already exist, as they do when they are reused across mixture requests. Charging their training to this experiment, $C_{\text{Policy-Merge}} \approx KC_{\text{RL}} + MC_{\text{eval}}(K)$ against $C_{\text{direct}} \approx MC_{\text{RL}}$ for K experts and M candidates: the six experts cost 166 GPU-hours, $6.7\times$ less than training the 40 candidates.

F On-Policy Distillation

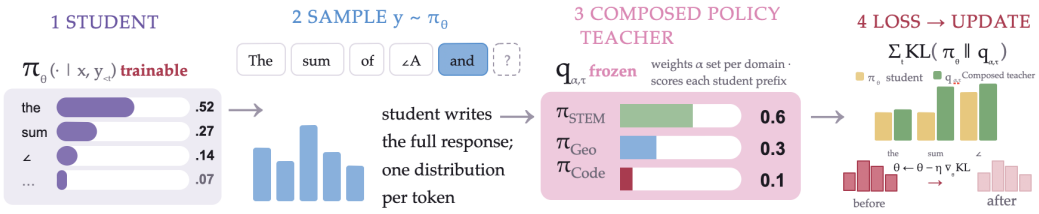


Figure 11: **Policy-Merge serves as a single teacher for on-policy distillation (OPD)**: The frozen composition $q_{\boldsymbol{\alpha},\tau}$ at $\boldsymbol{\alpha} = \mathbf{w}$ and $\tau = 1$, which is the teacher $q_{\mathbf{w}}$ of Eq. (6), scores each prefix the student samples. The student minimizes the per-token reverse KL to it.

F.1 OPD Objective and Estimator

Our estimator. Prefixes h_t are sampled from the current student at temperature 1 and then held fixed, so no gradient flows through their sampling. At every response position we compute the full-vocabulary reverse KL $\sum_{v\in\mathcal{V}} \pi_\theta(v \mid h_t)\big[\log \pi_\theta(v \mid h_t) - \log q(v \mid h_t)\big]$ exactly and differentiate it analytically. There is no sampled-token reward, advantage, baseline, discounting, importance ratio, clipping, or KL term to π_{ref}. The loss averages over the tokens of each response and over all responses of an update, and each rollout batch is used for a single optimizer step. The Policy-Merge teacher is $\log q_{\mathbf{w}} = \log \operatorname{softmax}\big(\sum_k w_k z_k\big)$ from the expert logits z_k, which equals Eq. (6). Weighted MOPD uses the target $\sum_k w_k \log \pi_k$, with each expert normalized but the combination not. Routed-expert MOPD uses the expert $\pi_{d(x)}$ of the prompt's domain.

Exactness. On a fixed prefix the two targets differ by $\log Z_{\mathbf{w}}(h_t)$, which does not depend on θ (Eq. (7)). The weighted-MOPD and Policy-Merge-teacher gradients are therefore identical for every sampled batch, not only in expectation, up to floating-point error. The identity is a statement about this per-prefix update: the sequence-level reverse KLs, differentiated through the prefix distribution, differ by $\mathbb{E}_{\pi_\theta}\big[\sum_t \log Z_{\mathbf{w}}(h_t)\big]$, whose gradient is nonzero in general. For a sampled-token estimator with per-token reward $\log q(y_t \mid h_t) - \log \pi_\theta(y_t \mid h_t)$ and no future credit, the two teachers agree in expectation, since $\mathbb{E}_{y_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(y_t \mid h_t)] = 0$, but not sample by sample. Summing rewards over future tokens, whitening advantages, or truncating the vocabulary would break even the equality in expectation.

Open-MOPD. Open-MOPD (Gao et al., 2026) routes each prompt to the teacher of its domain, so its target is $\pi_{d(x)}$, not a weighted combination of teachers. At each sampled prefix it takes

the student's top-16 tokens $\mathcal{S}_t$, gives each $v \in \mathcal{S}_t$ the advantage $\tilde{\pi}_\theta(v \mid h_t)\big[\log \pi_{d(x)}(v \mid h_t) - \log \pi_\theta(v \mid h_t)\big]$, where $\tilde{\pi}_\theta$ is the student renormalized over $\mathcal{S}_t$, and uses it directly in a clipped PPO objective, without returns or a critic. At ratio one this truncates the full-vocabulary reverse-KL gradient $\sum_v \pi_\theta(v \mid h_t)\big[\log \pi_\theta(v \mid h_t) - \log \pi_{d(x)}(v \mid h_t)\big]\nabla_\theta \log \pi_\theta(v \mid h_t)$ to $\mathcal{S}_t$, with $\tilde{\pi}_\theta$ in place of π_θ. The released student also reweights the loss per domain to balance token shares and to track the remaining teacher–student gap. Section 4.4 therefore compares a training-free equal-weight composition with a student distilled by domain-routed OPD. It compares two ways of consolidating the same teachers, not two distillation objectives. Our routed-expert MOPD arm (Table 5) uses Open-MOPD's choice of teacher with our full-vocabulary estimator.

F.2 Distillation Results

Table 5 pools the accuracies of Figure 7 over the 17 tasks and adds weight merging, a second composition that is not distilled.

Table 5: **On-policy distillation and the compositions it distils from**, measured against routed-expert MOPD. Each cell is accuracy averaged across all 17 Reasoning Gym tasks (3,400 items, binomial SE ≈ 0.84), with its gain over the routed-expert arm in parentheses.

	Training mixture			
	Alg 20 Gra 80	**Alg 50 Gra 50**	**Alg 90 Gra 10**	**Mean gain**
Compositions, not distilled				
Raw POLICY-MERGE (no OPD)	36.79 (−0.47)	38.68 (−0.67)	39.79 (+0.50)	−0.22
Weight merging	37.24 (−0.03)	38.29 (−1.06)	38.62 (−0.68)	−0.59
Distilled students				
POLICY-MERGE teacher	35.97 (−1.29)	37.85 (−1.50)	37.73 (−1.56)	−1.45
Routed expert MOPD (ref.)	37.26	39.35	39.29	reference

G Responses: Trained Mixture vs. Policy-Merge

Graphs · course schedule — Both correct · item #40 of 62 in this cell

PROMPT — gold answer: `False`

```
There are a total of 10 courses you have to take, labeled from 0 to 9.

You are given the following list of prerequisites, where prerequisites[i] = (a_i, b_i) indicates that you
      must first take course b_i if you want to take course a_i:
[(3, 6), (6, 2), (5, 7), (6, 7), (6, 4), (0, 9), (3, 8), (2, 8), (1, 7), (7, 9), (7, 6), (2, 0), (4, 2),
      (1, 9), (5, 9), (8, 7)]

Return True if you can finish all courses considering the prerequisites, or False otherwise.
Put ONLY your final answer inside <answer> </answer>.
```

Trained mixture correct 1,578 tokens

> Okay, let's try to figure out if all these courses can be taken based on their prerequisites. So, there are 10 courses labeled from 0 to 9. The prerequisites are given as a list of pairs (a_i, b_i). That means for each course a_i, you must take course b_i before you can take a_i.
>
> First, I need to check if there's any course that's not possible to take. Let me start by listing out all the prerequisites and see which ones are required.
>
> Looking at the list: [(3, 6), (6, 2), (5, 7), (6, 7), (6, 4), (0, 9), (3, 8), (2, 8), (1, 7), (7, 9), (7, 6), [... 4,148 characters elided ...]
>
> Course 7 is already in the loop.
>
> So course 1 can be taken after course 7.

Similarly, course 5 requires course 9.

Course 9 is in the loop.

Course 8 requires course 7.

So all courses are in this loop.

Therefore, there's no way to take all courses because of this cycle.

Hence, the answer should be False.
</think>

<answer>False</answer>

POLICY-MERGE composition correct 1,152 tokens

Okay, let's try to figure out if all the courses can be taken given their prerequisites. So, there are 10 courses labeled from 0 to 9. The prerequisites list is a bit long, so I need to check each course's dependencies.

First, I should probably list out all the courses and their prerequisites. Let me start by writing them down:

Courses: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Prerequisites:

(3, 6) −> course 6 depends on course 3

(6, 2) −> course 2 depends on course 6

(5, 7) −> course 7 depends on course 5

(6, 7) −> course 7 depends on course 6

(6, 4)
[... 2,828 characters elided ...]
3. Course 3 requires course 5. Course 5 requires course 1. Course 1 requires course 7. Course 7 requires course 8. Course 8 requires course 2. So course 2 is a cycle.

Yes, that's a cycle. Therefore, there's a cycle in the dependencies. Hence, it's impossible to finish all courses.

So the answer should be False.
</think>

<answer>False</answer>