TRAINING FOUNDATION MODELS WITH A SINGLE ALGORITHM

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation models are typically trained in stages: pretraining, midtraining, supervised fine-tuning (SFT), and reinforcement learning (RL), with changes in both data and learning algorithms. We show that this algorithmic separation is not fundamental. By deriving suitable RL objectives as maximum-likelihood fitting to model-generated target distributions, we place all of these stages under a single likelihood-based training algorithm. This perspective turns the foundation-model training lifecycle into a problem of choosing which data to train on, in what proportions, and when, with conventional staged training as a special case. We instantiate this framework in a unified training system that mixes static data and RL generations within a single trainer. Across controlled and real-world experiments, unified training improves the accuracy–compute tradeoff and task coverage while preserving broader capabilities, and provides a stronger starting point for continual learning. This work lays the groundwork for a unified science of foundation-model training, connecting mathematical principles, training infrastructure, and the development of model capabilities.

1 INTRODUCTION

Foundation models are conventionally trained through a sequence of stages: pretraining, midtraining, supervised fine-tuning (SFT), and reinforcement learning (RL) (DeepSeek-AI et al., 2025; Olmo et al., 2026). These stages have become embedded in specialized training recipes and infrastructure, and are often treated as separate learning problems. In this paper, we argue that this separation is not fundamental to the training process. Instead, the full training pipeline admits a strikingly simple view:

Foundation-model training can be viewed as a single learning objective driven by an evolving distribution of training data.

This shift changes the central design problem. Conventional pipelines organize design decisions around separate stages, each with its own training recipe. Under a unified view, the learning algorithm stays fixed from start to end, and the central design choices concern the evolving distribution of training data: which data the model learns from, in what proportions, and when. Pretraining text, demonstrations, and model generations selected with verifiable rewards become different data sources for the same underlying learning process. Conventional staged training is then one particular choice, in which the weight on each source abruptly shifts from one stage to the next.

The unified view also changes the scientific question. Our understanding of foundation-model training has historically developed one stage at a time: pretraining scaling laws characterize the relationship between data, compute, and model performance (Kaplan et al., 2020; Hoffmann et al., 2022), while a growing literature studies scaling during RL (Nimmaturi et al., 2026; Khatri et al., 2026; Tan et al., 2026). Research on transfer and interactions across stages shows that earlier training decisions also affect subsequent learning (Hernandez et al., 2021; Isik et al., 2025; Runwal et al., 2026; Shen et al., 2026). As a result, we know much less about how decisions made throughout training jointly determine the capabilities of the final model. A unified formulation lets us study these decisions together and judge them by their effect on the final model rather than by stage-specific metrics. This opens the door to a science of training trajectories: principles for how data and compute should evolve over the lifetime of a foundation model.

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

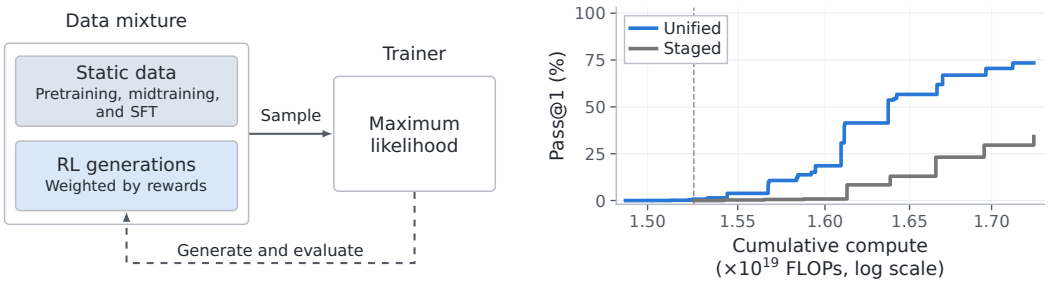


Figure 1: **Unified training framework. Left:** For suitable RL objectives, static data and RL generations share a weighted likelihood loss; the updated model supplies new generations for reward evaluation. **Right:** Unified training achieves higher accuracy within a fixed amount of compute. Each curve shows the best accuracy observed among checkpoints using at most the indicated compute: across unified runs with different data mixtures (blue), or along the staged training run (gray). Compute includes pretraining, midtraining, and RL. The axis begins at the staged pretraining budget; the vertical dashed line marks the end of staged midtraining. Figure 4 shows the full trajectories.

Making this view concrete requires bringing RL under the maximum-likelihood formulation already used for pretraining, midtraining, and SFT. Building on Maximum Likelihood Reinforcement Learning (MaxRL) (Tajwar et al., 2026), we extend its likelihood interpretation to arbitrary fixed generation policies and show how the resulting targets can be combined with static data in a single likelihood objective. This provides the theoretical foundation for *unified training*, a framework for learning from evolving mixtures of static and generated data under a common training rule, including when generation and optimization proceed independently. We implement this formulation through a common data representation, weighted likelihood loss, and trainer for both static and generated data. Figure 1 visualizes the unified training loop and shows that unified training can improve performance over staged training within the same total compute budget. Concretely, our contributions are as follows:

- **A unified formulation of foundation-model training.** We establish a common maximum likelihood formulation for pretraining, midtraining, SFT, and suitable RL objectives. This provides a theoretical basis for learning from changing mixtures of static and generated data, with staged training as a special case. The formulation also extends to other RL objectives and continuous rewards (Section 3), and it supports a unified training infrastructure (Section 4).
- **Empirical benefits across training settings.** Through controlled and real-world experiments, we demonstrate improvements of unified training over staged training in accuracy and coverage. We also show that unified training can preserve broader capabilities (Section 5).
- **Data mixing and compute allocation.** We characterize how mixing proportions and schedules shape the tradeoffs among accuracy, coverage, and retention, and study compute allocation across pretraining, midtraining, and RL (Section 5).

By placing data composition, timing, and compute allocation within a common framework, unified training provides a basis for developing principles that govern the full training process. We believe it has both scientific and practical value for studying foundation-model training as a whole. We discuss related work in Section A.

2 Preliminaries

We write $\pi_\theta(y \mid x)$ for the model's distribution of completions y given prompt x, with parameters θ.

Maximum likelihood estimation. Maximum likelihood estimation (MLE) fits the model to training examples by minimizing their negative log-likelihood. We represent each example as a prompt-completion pair (x, y), where the prompt may be empty. Let $\rho(x)$ denote the prompt distribution and $\mu(y \mid x)$ the conditional completion distribution in the training data. Their joint distribution is denoted by $\rho \circ \mu$, with $(\rho \circ \mu)(x, y) := \rho(x)\mu(y \mid x)$. MLE minimizes the loss

$$\mathcal{L}(\theta; \rho \circ \mu) := -\mathbb{E}_{x\sim\rho}\, \mathbb{E}_{y\sim\mu(\cdot|x)}[\log \pi_\theta(y \mid x)].$$

Pretraining, midtraining, and SFT all use this objective with different training data. Pretraining and midtraining fit text from general and curated corpora, respectively, using empty prompts; SFT fits

demonstrations paired with instructions. We refer to the fixed corpora and demonstrations used in these stages as *static data*.

For a fixed training data distribution, this loss differs from the expected forward KL divergence $\mathbb{E}_{x\sim\rho}[\mathrm{KL}(\mu(\cdot \mid x) \,\|\, \pi_\theta(\cdot \mid x))]$ only by a constant independent of θ. Both objectives therefore have the same gradient:

$$\nabla_\theta \mathcal{L}(\theta; \rho \circ \mu) = -\mathbb{E}_{x\sim\rho}\, \mathbb{E}_{y\sim\mu(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)].$$

Reinforcement learning. Reinforcement learning (RL) uses reward feedback to optimize model-generated completions. Given a prompt $x \sim \rho$, the model samples a completion $y \sim \pi_\theta(\cdot \mid x)$ and receives a reward $r(x, y) \in \{0, 1\}$.[1] The expected-reward objective is $J(\theta) := \mathbb{E}_{x\sim\rho}\, \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r(x, y)]$, with policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim\rho}\, \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x)]. \tag{1}$$

REINFORCE (Williams, 1992) estimates this expectation from sampled completions. Other methods, including PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), introduce choices of advantage estimation, reward normalization, and constraints on policy updates.

3 A Unified Likelihood View of Foundation-Model Training

Pretraining, midtraining, and SFT already share a common maximum-likelihood training rule, differing primarily in the data on which the model is trained. At first glance, RL seems to break this pattern, as it is typically formulated as policy optimization. To unify these stages, we build on the likelihood interpretation of MaxRL (Tajwar et al., 2026) to develop a common framework for these stages that also accommodates other generation policies, RL objectives, and continuous rewards.

Source	Prompts $\rho_{s,t}$	Target $\mu_{s,t}$
Pretraining	empty	μ_{pre}
Midtraining	empty	μ_{mid}
SFT	ρ_{sft}	μ_{sft}
RL	ρ_{rl}	$\pi^+_{\theta_t}$ or ν^+

Table 1: Data sources for the shared likelihood objective. For binary rewards, ν^+ is the success-conditioned distribution of the generation policy ν; $\nu = \pi_{\theta_t}$ recovers on-policy MaxRL.

3.1 MaxRL as Likelihood Fitting

For binary rewards, let $p_\theta(x) := \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r(x, y)]$ be the probability of success on prompt x, and let $\pi^+_\theta(y \mid x) := \pi_\theta(y \mid x, r = 1)$ be the success-conditioned distribution. Tajwar et al. (2026) show that the MaxRL gradient can be written as

$$\nabla_\theta \log p_\theta(x) = \mathbb{E}_{y\sim\pi^+_\theta(\cdot|x)}\left[\nabla_\theta \log \pi_\theta(y \mid x)\right].$$

Thus, at iteration t, a MaxRL update can be viewed as maximum-likelihood fitting to the current policy's success-conditioned distribution $\pi^+_{\theta_t}$, held fixed during the update:

$$\mathcal{L}(\theta; \rho \circ \pi^+_{\theta_t}) = -\mathbb{E}_{x\sim\rho}\mathbb{E}_{y\sim\pi^+_{\theta_t}(\cdot|x)}\left[\log \pi_\theta(y \mid x)\right].$$

This places RL in the same likelihood-fitting form as pretraining, midtraining, and SFT.

3.2 A Unified Likelihood Framework

Pretraining, midtraining, SFT, and MaxRL can therefore all be expressed as likelihood fitting. We represent each source s at iteration t by a prompt distribution $\rho_{s,t}$ and target distribution $\mu_{s,t}$, as summarized in Table 1.

Let $\lambda_{s,t} \geq 0$, with $\sum_s \lambda_{s,t} = 1$, denote the mixture weight of source s at iteration t. Together, the sources define a single joint distribution over training examples,

$$q_t := \sum_s \lambda_{s,t}(\rho_{s,t} \circ \mu_{s,t}).$$

[1] We begin with binary rewards for clarity. Section 3.2 discusses continuous rewards through TailRL (Ramasubramanian et al., 2026); Section G gives details.

Unified training applies the same likelihood loss to this mixture:

$$\mathcal{L}(\theta; q_t) = -\mathbb{E}_{(x,y)\sim q_t}[\log \pi_\theta(y \mid x)].$$

The source distributions and mixture weights are held fixed when differentiating the loss. Equivalently, one can sample source s with probability $\lambda_{s,t}$, draw an example from that source, and apply the same likelihood loss.

Both the source distributions and their mixture weights may evolve throughout training. The learning rule itself, however, remains unchanged: what changes is the distribution of data presented to the model. Conventional staged training is the special case in which all mixture weight is placed on one source at a time; unified training additionally allows sources to overlap and their proportions to vary throughout training.

The formulation above contains the core ingredients of unified training, and we turn to its implementation in the next section. Before doing so, we establish two extensions of the framework: it can incorporate generations produced by policies other than the model currently being optimized, and it can accommodate other policy-gradient objectives through different choices of target and effective prompt distributions.

Likelihood targets from arbitrary generators. The likelihood formulation does not require RL data to be generated by the model currently being optimized. Suppose completions are generated by a policy ν, held fixed while constructing the training target. Under binary rewards, its successful completions define

$$\nu^+(\cdot \mid x) := \nu(\cdot \mid x, r = 1).$$

The trainer can fit this distribution using the same maximum-likelihood loss,

$$\mathcal{L}(\theta; \rho \circ \nu^+) = -\mathbb{E}_{x\sim\rho}\mathbb{E}_{y\sim\nu^+(\cdot|x)}\left[\log \pi_\theta(y \mid x)\right].$$

When $\nu = \pi_{\theta_t}$, this recovers the on-policy MaxRL update of Section 3.1. For a different generator, the trainer instead fits that generator's success-conditioned distribution. Thus, generation and optimization can proceed independently while retaining the same likelihood training rule; the resulting update, however, is generally not the current-policy MaxRL gradient.

Other RL objectives. Under binary rewards, REINFORCE fits the same success-conditioned target $\pi^+_{\theta_t}$ as MaxRL, but uses the effective prompt distribution

$$\rho_{\mathrm{RF}}(x) \propto \rho(x) p_{\theta_t}(x).$$

This gives greater weight to prompts with higher success probability, whereas MaxRL retains ρ. The framework also accommodates rewards normalized to $[0, 1]$, including continuous-reward REINFORCE and TailRL (Ramasubramanian et al., 2026). Section G gives the corresponding targets, prompt distributions, and derivations.

4 Practical Implementation

Building on the theoretical foundation in Section 3.2, which allows static data and RL generations to be combined under a common likelihood objective, we implement a unified training system with a shared data format, loss, and trainer. A generator supplies RL generations and their weights to a shared data buffer, which also holds static data. Generation and optimization run asynchronously, with periodic synchronization of the model weights (Figure 2).

Unified loss and data format. Each training example is a pair (z, w), where $z = (z_1, \ldots, z_L)$ is a token sequence and $w \in \mathbb{R}^L_{\geq 0}$ specifies the weights on its token negative log-likelihoods. Pretraining and midtraining use unit weights on corpus tokens; SFT uses unit weights on demonstration tokens and zero weights on prompt tokens. RL uses the same prompt mask and assigns each completion a response weight A, computed from its reward and the sampled pass rate. These assignments give a common data format for all sources (Figure 2, right). In practice, we use a token mean over a batch B:

$$\hat{\mathcal{L}}(\theta; B) := -\frac{1}{T_B} \sum_{(z,w)\in B} \sum_{\ell=1}^{L} w_\ell \log \pi_\theta(z_\ell \mid z_{<\ell}), \tag{2}$$

where T_B counts the total number of non-prompt tokens in the batch.

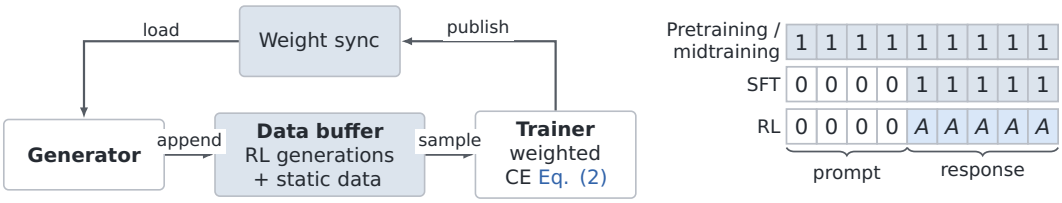


Figure 2: **Left:** the training system. The generator produces RL data from a recent copy of the weights and appends it to the data buffer, which also holds static data; the trainer samples from the buffer and periodically publishes its weights, which the generator loads. **Right:** the data format. Each cell is a token and the number is its weight in Eq. (2); the stages differ only in the weights.

Data generation. The generator samples N completions $y_1, \ldots, y_N \sim \pi_{\theta_s}(\cdot \mid x)$ per prompt x, where π_{θ_s} is a recent copy of the model. For MaxRL with binary rewards, it estimates the pass rate by $\hat{p} := \frac{1}{N}\sum_i r(x, y_i)$. Each correct completion receives weight $A := 1/\hat{p}$ on its response tokens; incorrect completions receive zero (Tajwar et al., 2026).

Data mixing and asynchronous training. The trainer implements the data mixture by sampling static data and RL generations from a shared buffer. For a target token fraction f_s from source $s \in S$, examples are sampled at a rate proportional to f_s/T_s, where T_s is the mean number of corpus or response tokens per example. The mixture can also be specified by the number of examples from each source. Section B.1 collects the mixing and loss-weighting conventions used in the experiments.

Generation and optimization proceed asynchronously: generators append completed batches to the buffer, while the trainer samples from it and updates the model. The trainer publishes its weights every k updates, and generators load them between batches. Section D.6 reports the resulting policy staleness and compares training throughput.

5 Experiments

Our experiments assess the practical benefits of unified training and its value for studying the full training process. We examine how data composition and timing affect model capabilities, the use of data and compute, and subsequent learning:

1. **Performance and forgetting.** How do unified and staged training compare in accuracy, coverage, and knowledge retention?
2. **Data reuse.** Can unified training better use additional compute on a fixed corpus?
3. **Data mixing.** How do data proportions and timing affect intermediate and final performance?
4. **Compute allocation.** Do the gains persist after accounting for all training compute?
5. **Continual learning.** Can unified training improve the starting point for continual learning?

We first compare unified and staged training across the full pipeline of pretraining, midtraining, and RL, in a controlled setting (Section 5.1). Then, with real-world heterogeneous midtraining data, we study the resulting capability tradeoffs, how scheduling can alter them, and their consequences for continual learning (Section 5.2). Unless otherwise specified, staged baselines use on-policy MaxRL (Tajwar et al., 2026) for RL; Section F documents the GRPO instabilities that motivate this choice.

5.1 Full Unified Training under Controlled Reasoning Tasks

Experimental setup. To control data exposure throughout training, we train a 124M-parameter model from scratch, retaining the same 20B-token pretraining stream and fixed midtraining corpus across runs. We use number-assignment puzzles in the style of Reasoning Gym (Stojanovski et al., 2025), which provide verifiable solutions and controllable difficulty. This allows us to use directly task-relevant midtraining data and control whether the RL tasks have the same or greater difficulty.

Concretely, we procedurally generate chains of thought and answers for a subset of the questions at Levels 1–4 to form the midtraining corpus. RL runs train and evaluate on Levels 4–6. Level 4 tests learning at a difficulty represented in the midtraining corpus, while Levels 5–6 test adaptation through RL to harder problems.

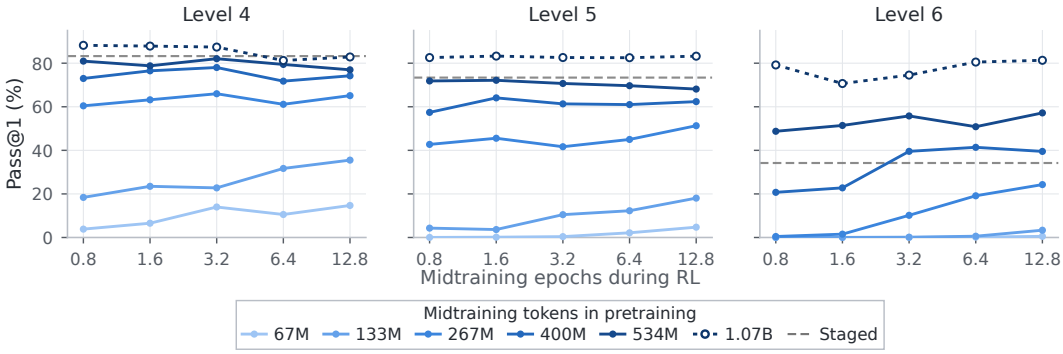


Figure 3: **Data reuse across pretraining and RL.** Final pass@1 after 4000 RL updates. Colors vary midtraining exposure during pretraining; the horizontal axis varies passes over the Level 4 corpus during RL. Both increase exposure to fixed data. Gray dashed lines show the selected staged reference. Each panel trains and evaluates at the indicated difficulty; the dotted 1.07B-token curves mark the largest mixture, whose pre-RL retention cost is shown in Figure 5.

To study data reuse and compute scaling, we construct a large midtraining corpus but allow multiple passes over it in both unified and staged training. All runs in this section use a fixed budget of 4000 RL updates, so that RL compute is matched. Further setup details appear in Section C.

5.1.1 Final Performance and Data Reuse

We first ask how final performance depends on scaling midtraining data exposure during pretraining and during RL (RQs 1 to 3). At each RL-phase midtraining exposure shown in Figure 3, increasing midtraining exposure during pretraining improves final accuracy on all three levels. Increasing midtraining exposure during RL has a different pattern: it generally improves final performance when exposure during pretraining is relatively low, but the trend is less consistent when exposure during pretraining is high.

Concretely, with 1.07B midtraining tokens during pretraining (the most midtraining data reuse), the best unified configuration outperforms staged training on all three levels, while with 534M midtraining tokens during pretraining, the best unified configuration still outperforms staged training on the hardest task level, but only matches staged training on the two easier levels. Thus, scaling midtraining data mixing into both pretraining and RL can improve final performance, even on problems harder than those in the midtraining corpus, but the best mixture strategy depends on the interaction between the two phases.

5.1.2 Compute Comparison

The previous section compares final performance without taking the total compute cost into account, which is important for practical deployment and for understanding the tradeoffs between staged and unified training. In Figure 4, we plot accuracy throughout RL against cumulative compute across the full pipeline for a compute-matched comparison (RQ4). Each colored curve follows one unified run: darker colors indicate more midtraining exposure during pretraining, while curves of the same color vary the midtraining mixture during RL. The starting points include the compute spent before RL, and the gray curves show staged training, selected by mean final RL accuracy across the three levels (Section C.6). The "best observed checkpoints" curve identifies the highest observed accuracy within each compute budget, allowing us to compare different mixtures and training durations under the same compute budget.

On Levels 4–5, staged training starts RL with higher accuracy and remains competitive over part of the compute range, but unified training with the largest pretraining mixture subsequently achieves higher accuracy. The advantage of unified training is more prominent on the harder task: unified training defines the entire observed frontier on Level 6, with a larger margin over staged training. Extending the comparison to five staged midtraining budgets gives staged training a larger role on the frontier at lower compute, including on Level 6, while unified training retains the highest accuracy at larger budgets (Section C.7). Together, these results show that unified training can outperform

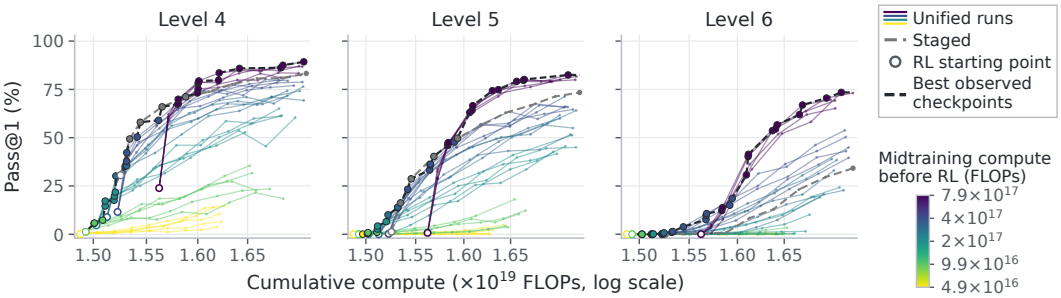


Figure 4: **Unified training defines most of the observed accuracy–compute frontier.** Colored curves show unified runs; gray dashed curves show the 32-epoch staged reference. Open circles mark RL starting points, and colored or gray markers identify the source of each checkpoint on the black frontier. Each column ends at the staged run's final compute budget. Compute includes pretraining, midtraining, and RL. Section B.2 gives compute calculations.

staged training outside the low-compute regime. The best data mixing strategy is determined by the compute budget, task difficulty, and relevance of the midtraining data to the RL task.

5.1.3 Pretraining Retention

We next ask whether the reasoning gains from increasing midtraining exposure during pretraining come at the expense of general capabilities (RQ1). In Figure 5, we evaluate held-out pretraining loss and mean performance on five general benchmarks before RL. Both measures remain close to the pretraining-only reference through 534M midtraining tokens. At the largest mixture, held-out loss rises more sharply and mean benchmark performance decreases further.

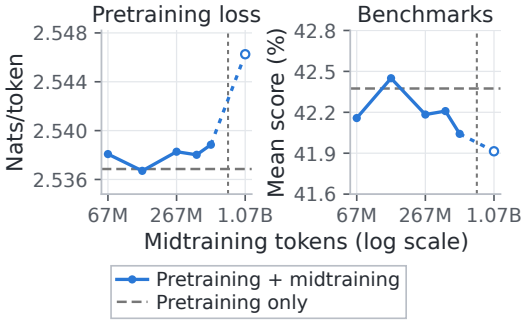


Figure 5: **Midtraining exposure and pretraining performance.** Mean score averages five benchmarks. Dashed lines mark pretraining alone; vertical lines separate the largest mixture, whose higher loss motivates its dotted styling.

Concretely, the 534M mixture improves final accuracy over staged training on Level 6 while retaining similar pre-RL performance on these general-capability measures. The largest mixture achieves the highest final reasoning accuracy, but with a greater retention cost before RL. Thus, the preferred midtraining mixture depends on how final task performance is balanced against retention of general capabilities.

5.1.4 Loss Aggregation Comparison

We now study how the choice of loss affects learning from the same mixture of midtraining and RL data. Unified training allows their relative contributions to change with the realized token mixture, whereas the alternative combines separately normalized losses using a fixed coefficient. With initialization, midtraining exposure, and RL rollout budget held fixed, we compare the performance of unified training and separate loss on the same reasoning RL tasks. We defer the full results to Section C.9, where we show that using separate midtraining and RL losses can be competitive with unified training under certain conditions. The best observed performance depends on the coefficient, whose preferred value varies with the task, initialization, and mixture. Thus, the unified training loss is a natural and competitive approach for combining different sources of data.

5.2 Unified Training with Real-world Midtraining Data

The controlled setting uses midtraining data that is task-specific and RL tasks whose difficulties can be precisely controlled. We now use a heterogeneous midtraining corpus to examine how unified training during mathematical RL affects task performance, broader capabilities, and continual learning. We use a 0.76B-parameter model that we train from scratch through pretraining and midtraining. Unified training starts from this base model, trains on Math12K, and mixes data from

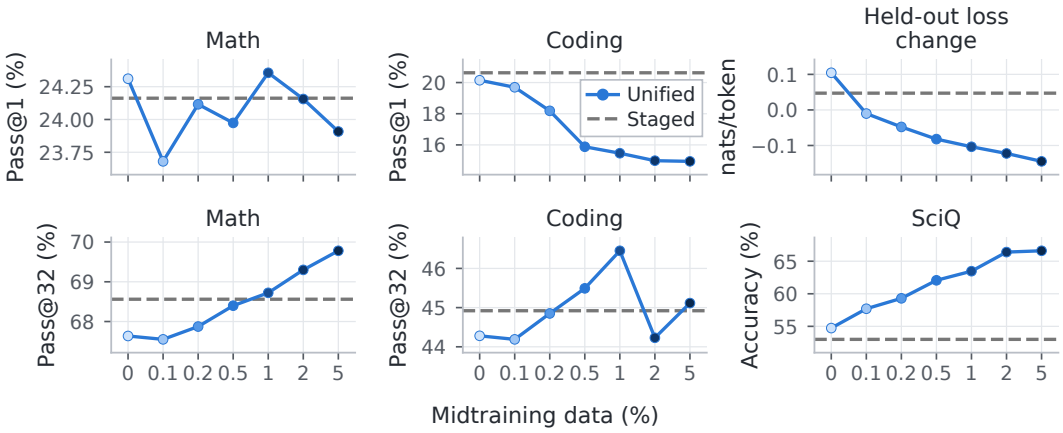


Figure 6: **Effect of the midtraining data proportion.** Final results after 12000 updates. Token proportions are estimated before training and shown categorically; loss changes are relative to the base model.

a 240M-token corpus with the same composition as the midtraining corpus, including web text, mathematics, code, and reasoning traces. Staged training uses the additional corpus before RL. Section D.1 gives the setup. We report *Math* as the average of MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2024) and GSM8K (Cobbe et al., 2021), and *Coding* as the average of HumanEval+ and MBPP+ (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023), with equal benchmark weights.

5.2.1 Data Proportions and Performance Tradeoffs

We first ask how increasing midtraining exposure affects the balance between mathematical learning and retention (RQ1 and RQ3). In Figure 6, we vary the constant midtraining proportion from 0 to 5% and evaluate both mathematical performance and capabilities beyond the RL task. The 0% control uses the same asynchronous trainer and reward weighting as the other unified runs, allowing us to assess the effect of adding midtraining data with the implementation held fixed. Section D.6 compares this control with staged RL.

Increasing the midtraining proportion generally improves Math coverage while maintaining similar Math accuracy. It also improves retention: held-out midtraining loss falls below the base-model value, while it increases under staged training; SciQ (Welbl et al., 2017) accuracy also improves with additional mixing. However, Coding accuracy falls below both the 0% control and staged training, although we also observe an increase in Coding coverage. Thus, continued learning from heterogeneous midtraining data can improve mathematical coverage and knowledge retention together, but yields smaller gains in Coding accuracy over the base model at low mixing proportions and declines at higher proportions. Section D.2 reports the full benchmark results.

5.2.2 Data Mixing Schedules

The preceding comparison varies how much midtraining data is used, but keeps its proportion constant throughout RL. We next ask whether changing when this data is used can reduce the coding accuracy cost while preserving the retention benefits (RQ3). In Figure 7, we compare constant mixing, linear decay, linear growth, and decay to zero halfway through training, all with an average configured midtraining proportion of 0.5% (Section B.1).

When mixing ends halfway through training, Coding accuracy improves and SciQ gains persist, while held-out corpus loss rises. Compared with constant mixing, this schedule achieves higher final Coding and SciQ accuracy but retains less of the reduction in corpus loss. Math accuracy and coverage remain similar across the four schedules (Section D.3). Thus, the preferred schedule depends on how performance across tasks is balanced; we leave a detailed analysis of these interactions to future work.

5.2.3 Continual Learning

We next address the final research question on continual learning (RQ5), asking whether unified training produces models that perform better in subsequent learning. In Figure 8, we apply the same

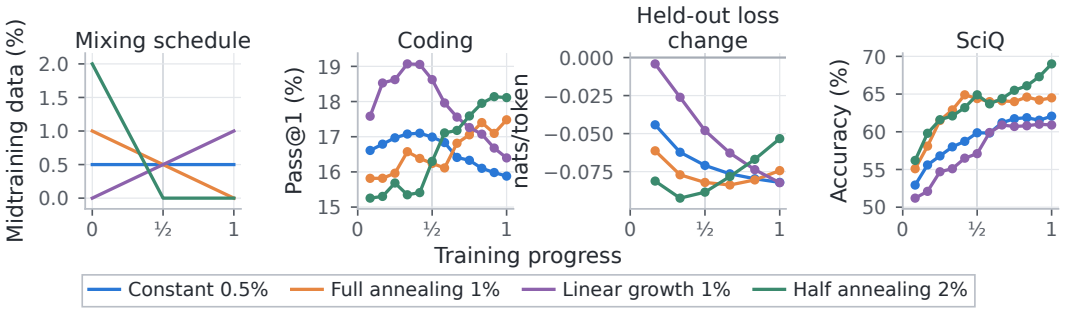


Figure 7: **Effect of the data mixing schedule.** All schedules have an average configured midtraining proportion of 0.5%. Full and half annealing reach zero at the end and midpoint, respectively.

coding RL procedure, without midtraining data, to the staged mathematical model and unified models trained with constant midtraining proportions of 0.5, 1, 2, and 5%. More details of the experimental setup can be found in Section D.7.

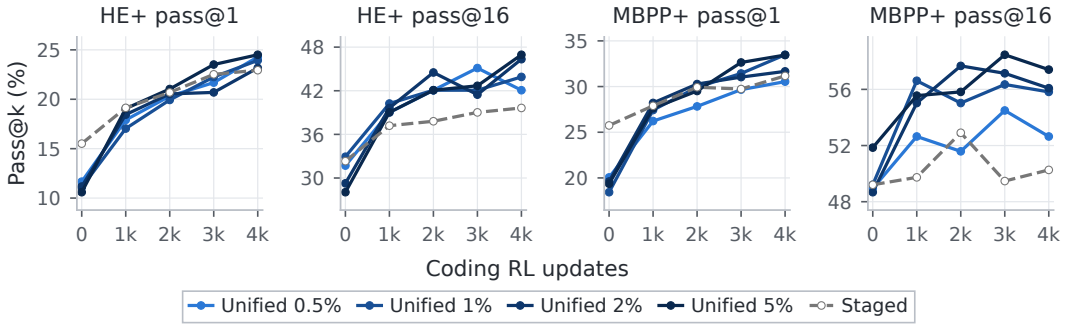


Figure 8: **Coding RL after mathematical training.** Percentages identify the midtraining proportions during mathematical training; all models then use coding RL without midtraining data. HE+ denotes HumanEval+; additional evaluations appear in Section D.7.

Despite lower initial coding accuracy, the unified models generally achieve higher accuracy and coverage than staged training after subsequent coding RL, with the only exception being MBPP+ pass@1 at 0.5% mixing. This suggests that unified training provides a better base for continual learning than staged training, especially when the midtraining proportion is sufficiently high during unified training. The finding is further verified in a continual learning setting in a general reasoning domain with WebInstruct (Ma et al., 2026) (Section D.8).

5.2.4 Additional Analyses

We further examine the roles of the RL objective, the relationship between unified training and policy regularization, and corpus composition. The comparison between MaxRL and REINFORCE in unified training is shown in Section D.5. In Section E, we show the role of midtraining data in RL during unified training is fundamentally different from explicit policy regularization. Finally, we investigate the impact of different midtraining data compositions for unified training in Section D.4.

6 Discussion

We discuss several limitations and future directions of our work. Our experiments use relatively small models and mostly optimize one RL task at a time. Extending the study to larger models and joint multitask training would test how the observed benefits depend on scale and interactions among tasks. Our results reveal trends, patterns and tradeoffs among accuracy, coverage, and retention, but do not establish scaling laws or a systematic rule for selecting data mixtures and schedules. Studies across model sizes and data budgets could guide how compute is allocated between reusing static data and generating new responses, and how this allocation should evolve during training. Our formulation also does not fully eliminate the notion of stages, since downstream applications of foundation models may still require a separate adaptation and fine-tuning stage.

AI USE STATEMENT

In this work, we used generative AI tools to assist with experimental code, data processing, figure preparation, literature searches, reference checks, and manuscript drafting and revision. The authors reviewed and revised the AI-assisted text, code, and analyses and take responsibility for the final manuscript, results, and associated artifacts.

ETHICS STATEMENT

This work studies foundation-model training using existing text corpora, procedurally generated tasks, and established benchmarks. More efficient training and stronger reasoning or coding capabilities may support beneficial applications, but can also amplify the risks associated with language models and their training data. Our evaluations measure task performance and retention; they do not establish safety, fairness, or privacy guarantees. Downstream applications therefore require evaluation appropriate to their use and data.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the formulation and implementation, with additional derivations in Section G. Sections B.1 and B.2 specify the data mixtures and compute calculations. Sections C and D.1 document the datasets, training hyperparameters, model selection, and evaluation protocols for the two experimental settings; Section E provides the KL-regularization comparison. We will release the code, data, and model checkpoints to support reproducibility and future research upon publication.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1ANxQW0b.

Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. OpenCodeInstruct: A large-scale instruction tuning dataset for code LLMs. *arXiv preprint arXiv:2504.04030*, 2025. URL https://arxiv.org/abs/2504.04030.

Loubna Ben allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan Son NGUYEN, Ben Burtenshaw, Clémentine Fourrier, Haojun Zhao, Hugo Larcher, Mathieu Morlon, Cyril Zakka, Colin Raffel, Leandro Von Werra, and Thomas Wolf. SmolLM2: When smol goes big — data-centric training of a fully open small language model. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=3JiCl2A14H.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Christina Baek, Ricardo Pio Monti, David Schwab, Amro Abbas, Rishabh Adiga, Cody Blakeney, Maximilian Böther, Paul Burstein, Aldo Gael Carranza, Alvin Deng, et al. The finetuner's fallacy: When to pretrain with your finetuning data. In *Conference on Language Modeling (COLM)*, 2026.

BigScience Workshop. BLOOM: A 176b-parameter open-access multilingual language model. *Journal of Machine Learning Research*, 25(422):1–74, 2024. URL http://jmlr.org/papers/v25/23-0581.html.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021. URL https://arxiv.org/abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. URL http://jmlr.org/papers/v24/22-1144.html.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. 2021. URL https://arxiv.org/abs/2110.14168.

Peter Dayan and Geoffrey E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, February 1997. ISSN 1530-888X. doi: 10.1162/neco.1997.9.2.271. URL http://dx.doi.org/10.1162/neco.1997.9.2.271.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, et al. Nemotron-climb: Clustering-based iterative data mixture

bootstrapping for language model pre-training. *Advances in Neural Information Processing Systems*, 38, 2026.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=m7p5O7zblY.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2026a. URL https://arxiv.org/abs/2505.24298.

Yuqian Fu, Tinghong Chen, Jiajun Chai, Xihuai Wang, Songjun Tu, Guojun Yin, Wei Lin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. Srft: A single-stage method with supervised and reinforcement fine-tuning for reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 102430–102459, 2026b.

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, 2024. URL https://arxiv.org/abs/2407.21783.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. OLMo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15789–15809, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.841. URL https://aclanthology.org/2024.acl-long.841/.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8342–8360, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL https://aclanthology.org/2020.acl-main.740/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer, 2021. URL https://arxiv.org/abs/2102.01293.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep Q-learning from demonstrations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2159-5399. doi: 10.1609/aaai.v32i1.11757. URL http://dx.doi.org/10.1609/aaai.v32i1.11757.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. doi: 10.52202/068431-2176. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018. URL https://arxiv.org/abs/1803.00933.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. Scaling laws for downstream task performance in machine translation. In *International Conference on Learning Representations*, volume 2025, pp. 88769–88790, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/dce0ad3bd4981fea9a5a5a274a2256d9-Paper-Conference.pdf.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Venkata Sai Surya Subramanyam Duvvuri, Manzil Zaheer, Inderjit Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms. In *International Conference on Learning Representations*, volume 2026, pp. 72438–72467, 2026.

Suhas Kotha and Percy Liang. Replaying pre-training data improves fine-tuning, 2026. URL https://arxiv.org/abs/2603.04964.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tülu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic Q-Ensemble. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1702–1712. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/lee22d.html.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018. URL https://arxiv.org/abs/1805.00909.

Haoyang Li, Sheng Lin, Fangcheng Fu, Yuming Zhou, Xiaodong Ji, Yanfeng Zhao, Lefeng Wang, Jie Jiang, and Bin Cui. Staleflow: Staleness-aware data management for mitigating data skewness in fully disaggregated rl post-training, 2026. URL https://arxiv.org/abs/2601.12784.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL https://aclanthology.org/2022.acl-long.229/.

Emmy Liu, Graham Neubig, and Chenyan Xiong. Midtraining bridges pretraining and posttraining distributions. In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Mingyang Liu, Gabriele Farina, and Asuman Ozdaglar. UFT: Unifying supervised and reinforcement fine-tuning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 101347–101383. Curran Associates, Inc., 2025. doi: 10.52202/085713-3394. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/92e886487a8354b03d8bf4416eae6d7d-Paper-Conference.pdf.

Xingtai Lv, Yuxin Zuo, Youbang Sun, Hongyi Liu, Yuntian Wei, Zhekai Chen, Xuekai Zhu, Kaiyan Zhang, Bingning Wang, Ning Ding, and Bowen Zhou. Towards a unified view of large language model post-training, 2026. URL https://arxiv.org/abs/2509.04419.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. *Advances in Neural Information Processing Systems*, 38:56596–56618, 2026.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 50358–50376. Curran Associates, Inc., 2023. doi: 10.52202/075280-2191. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/9d89448b63ce1e2e8dc7af72c984c196-Paper-Conference.pdf.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299. IEEE, May 2018. doi: 10.1109/icra.2018.8463162. URL http://dx.doi.org/10.1109/ICRA.2018.8463162.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021. URL https://arxiv.org/abs/2006.09359.

Datta Nimmaturi, Vaishnavi Bhargava, Rajat Ghosh, Johnu George, and Debojyoti Dutta. Predictive scaling laws for efficient GRPO training of large reasoning models, 2026. URL https://arxiv.org/abs/2507.18014.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg,

Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2026. URL https://arxiv.org/abs/2512.13961.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, et al. GPT-4 technical report, 2023. URL https://arxiv.org/abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. doi: 10.52202/068431-2011. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pp. 745–750, 2007.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):1607–1612, July 2010. ISSN 2159-5399. doi: 10.1609/aaai.v24i1.7727. URL http://dx.doi.org/10.1609/aaai.v24i1.7727.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation, 2025. URL https://arxiv.org/abs/2509.19128.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.049.

Shrinivas Ramasubramanian, Daman Arora, Fahim Tajwar, Guanning Zeng, Qingyang Wu, Zhongzhu Zhou, Chenfeng Xu, Haiwen Feng, Yuda Song, Aarti Singh, Ruslan Salakhutdinov, J. Andrew Bagnell, Jeff Schneider, and Andrea Zanette. Tail-likelihood reinforcement learning. 2026. URL https://arxiv.org/abs/2609.02987.

Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryenvpEKDr.

Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

Bharat Runwal, Ashish Sunil Agrawal, Anurag Roy, and Rameswar Panda. PRISM: Demystifying retention and interaction in mid-training. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=giNBsVVFGt.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, April 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6399. URL http://dx.doi.org/10.1609/aaai.v34i05.6399.

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

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

Jingyan Shen, Ang Li, Salman Rahman, Yifan Sun, Micah Goldblum, Matus Telgarsky, and Pavel Izmailov. Understanding reasoning from pretraining to post-training, 2026. URL https://arxiv.org/abs/2607.16097.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Yuda Song, Yifei Zhou, Ayush Sekhari, J. Andrew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using both offline and online data can make RL efficient. In *International Conference on Learning Representations*, April 2023. URL https://openreview.net/forum?id=yyBis80iUuU.

Yuda Song, Drew Bagnell, and Aarti Singh. Hybrid reinforcement learning from offline observation alone. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 46019–46049. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/song24a.html.

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. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-1913. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/528d56195a2c77c808494c86fa7c77ad-Paper-Datasets_and_Benchmarks_Track.pdf.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026. URL https://icml.cc/virtual/2026/poster/65332.

Zelin Tan, Hejia Geng, Xiaohang Yu, Mulei Zhang, Guancheng Wan, Yifan Zhou, Qiang He, Xiangyuan Xue, Heng Zhou, Yutao Fan, et al. Scaling behaviors of LLM reinforcement learning post-training: An empirical study in mathematical reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31300–31319, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1444. URL https://aclanthology.org/2026.acl-long.1444/.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Andrew Wagenmaker and Aldo Pacchiano. Leveraging offline data in online reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35300–35338. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/wagenmaker23a.html.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems*, volume 36, pp. 69798–69818. Curran Associates, Inc., 2023. doi: 10.52202/075280-3059. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/dcba6be91359358c2355cd920da3fcbd-Paper-Conference.pdf.

Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 27395–27407. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/e61eaa38aed621dd776d0e67cfeee366-Paper.pdf.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, volume 38, pp. 117157–117186. Curran Associates, Inc., 2025. doi: 10.52202/085713-3904. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a9d5c33e38442450f9781bd7bad3c81e-Paper-Conference.pdf.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Jun Zhan, Yunhua Zhou, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. In *International Conference on Learning Representations*, volume 2025, pp. 82263–82287, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/cc84bfabe6389d8883fc2071c848f62a-Paper-Conference.pdf.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=TBaUfO9znF.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, and Jingren Zhou. On-policy rl meets off-policy experts: Harmonizing supervised fine-tuning and reinforcement learning via dynamic weighting. In *International Conference on Learning Representations*, volume 2026, pp. 120693–120726, 2026b.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, Hongyu Zhou, Yimin Jiang, Yibo Zhu, and Daxin Jiang. Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation, 2025. URL https://arxiv.org/abs/2504.15930.

Yifei Zhou, Ayush Sekhari, Yuda Song, and Wen Sun. Offline data enhanced on-policy policy gradient with provable guarantees. In *International Conference on Learning Representations*,

volume 2024, pp. 35725–35760, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/9949e6906be6448230cdba9a4cb2d564-Paper-Conference.pdf.

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

Appendix Contents

A Related Work **20**
- A.1 Foundation-Model Training . . . 20
- A.2 Understanding Foundation-Model Training . . . 20
- A.3 Hybrid Offline and Online Reinforcement Learning . . . 20
- A.4 Connections between Reinforcement Learning and Supervised Learning . . . 21
- A.5 Asynchronous Reinforcement Learning . . . 21

B Data Mixing and Compute Calculation **22**
- B.1 Data Mixing . . . 22
- B.2 Compute Calculations . . . 23

C Pretraining, Midtraining, and RL: Setup and Additional Results **25**
- C.1 Tasks and Midtraining Data . . . 25
- C.2 Model and Pretraining . . . 26
- C.3 Staged Initialization and RL Training . . . 26
- C.4 Evaluation Protocol . . . 27
- C.5 Reasoning Performance before RL . . . 28
- C.6 Staged Midtraining Sweep . . . 29
- C.7 Accuracy and Compute across Midtraining Budgets . . . 30
- C.8 Retaining Pretraining Data during RL . . . 31
- C.9 Combining Midtraining and RL Losses . . . 32

D Details of Training with Real-world Midtraining Data **35**
- D.1 Experimental Setup . . . 35
- D.2 Effect of the Midtraining Data Proportion . . . 36
- D.3 Data Mixing Schedules . . . 39
- D.4 Effect of Code Data . . . 41
- D.5 MaxRL versus REINFORCE . . . 42
- D.6 Unified Training with Zero Mixing versus Staged RL . . . 44
- D.7 Coding RL after Mathematical Training . . . 45
- D.8 WebInstruct RL after Mathematical Training . . . 47

E Comparison with KL Regularization **48**

F Examples of GRPO Collapse **50**

G Policy Gradient Objectives and Continuous Rewards **51**
- G.1 REINFORCE . . . 51
- G.2 Continuous Rewards with TailRL . . . 52

A Related Work

A.1 Foundation-Model Training

Large-scale maximum likelihood pretraining established the basis for broadly capable language models (Brown et al., 2020; Zhang et al., 2022; Chowdhery et al., 2023; BigScience Workshop, 2024). Subsequent systems combine this foundation with supervised learning, preference optimization, and RL through increasingly elaborate training pipelines (OpenAI et al., 2023; Touvron et al., 2023; Grattafiori et al., 2024; DeepSeek-AI et al., 2025; Guo et al., 2025). Open research efforts have made training data, recipes, and intermediate models more accessible, supporting closer examination of data curation, mixture schedules, and post-training procedures (Groeneveld et al., 2024; allal et al., 2025; Lambert et al., 2025; Olmo et al., 2026). Our work studies the organization of this training process through a common formulation and implementation for static and generated data, allowing their composition and timing to vary without requiring a separate training algorithm for each stage.

A.2 Understanding Foundation-Model Training

Empirical scaling studies characterize how performance varies with model size, data, and compute (Hestness et al., 2017; Rosenfeld et al., 2020; Kaplan et al., 2020; Hoffmann et al., 2022). Subsequent work examines the value of repeated data when fresh data are limited (Muennighoff et al., 2023) and the scaling of RL post-training, where initialization, model size, and optimization choices affect learning efficiency (Nimmaturi et al., 2026; Khatri et al., 2026; Tan et al., 2026).

Understanding the full training process also requires examining how earlier training shapes subsequent learning. Studies of transfer scaling and domain adaptation characterize how downstream performance depends on preceding training data (Gururangan et al., 2020; Hernandez et al., 2021; Isik et al., 2025), while recent reasoning studies examine how pretraining and midtraining affect subsequent RL and capability retention (Zhang et al., 2026a; Runwal et al., 2026; Shen et al., 2026). Complementary work optimizes data mixtures and studies the timing of domain exposure and the benefits of later replay (Xie et al., 2023; Ye et al., 2025; Liu et al., 2026; Baek et al., 2026; Kotha & Liang, 2026). Our work complements these empirical accounts with a common likelihood formulation that accommodates both static and RL-generated data, enabling the composition, timing, and reuse of these sources to be studied across the training process.

A.3 Hybrid Offline and Online Reinforcement Learning

Combining demonstrations or other prior experience with online interaction has long been used to improve exploration and sample efficiency in RL (Ross & Bagnell, 2012; Hester et al., 2018; Nair et al., 2018; Rajeswaran et al., 2018). This principle underlies a substantial literature on offline-to-online and hybrid RL, including practical algorithms for incorporating prior data into online learning (Nair et al., 2021; Lee et al., 2022) and theoretical analyses of when access to both sources improves learning efficiency (Xie et al., 2021; Song et al., 2023; Wagenmaker & Pacchiano, 2023; Zhou et al., 2024; Song et al., 2024).

Language-model post-training likewise combines static corpora or demonstrations with online RL. InstructGPT adds a pretraining likelihood term to the PPO objective to reduce regressions on public benchmarks (Ouyang et al., 2022), and recent work provides growing evidence that combining demonstrations with RL can improve learning and preserve capabilities (Yan et al., 2025; Liu et al., 2025; Fu et al., 2026b; Zhang et al., 2026b). These methods typically add a separately weighted supervised term to the RL loss, with a coefficient that is tuned or scheduled (Ouyang et al., 2022; Zhang et al., 2026b), or adapt the RL objective to off-policy data (Yan et al., 2025). In unified training, both sources share one likelihood objective, and their relative weight is set by the data mixture; Figure 13 compares this with a separately weighted supervised loss. Our work further studies the joint use of these sources across the full foundation-model training process, focusing on how their composition, timing, and reuse affect learning across pretraining, midtraining, and RL.

A.4 Connections between Reinforcement Learning and Supervised Learning

Connections between RL, supervised learning, and probabilistic inference have long motivated methods based on expectation-maximization, weighted regression, and constrained policy fitting (Dayan & Hinton, 1997; Peters & Schaal, 2007; Peters et al., 2010; Abdolmaleki et al., 2018; Levine, 2018; Peng et al., 2019). Reward-guided self-training methods such as STaR, ReST-EM, and RAFT iteratively train models by maximum likelihood on selected generated responses (Zelikman et al., 2022; Gulcehre et al., 2023; Singh et al., 2024; Dong et al., 2023). Related work also develops unified analyses of language-model post-training (Lv et al., 2026). Our formulation builds directly on the likelihood objectives of MaxRL and TailRL (Tajwar et al., 2026; Ramasubramanian et al., 2026). We use this connection to place static data and suitable RL targets within a common likelihood formulation, justifying mixtures of their training distributions and supporting a shared data format, loss, and trainer.

A.5 Asynchronous Reinforcement Learning

Our practical implementation is related to asynchronous RL, which separates acting from learning to make use of parallel hardware (Mnih et al., 2016). Distributed actor-learner systems build on this separation by passing experience from many actors to a central learner through a replay buffer or a trajectory queue (Horgan et al., 2018; Espeholt et al., 2018; 2019). RL systems for language models follow the same design, running generation and training on separate resources connected by a buffer or a stream (Zhu et al., 2025; Zhong et al., 2025). These systems differ mainly in how they limit policy lag: AReaL and StaleFlow bound the number of policy versions between generation and training (Fu et al., 2026a; Li et al., 2026), whereas PipelineRL updates the generator weights during generation (Piché et al., 2025).

B Data Mixing and Compute Calculation

Section B.1 collects the data mixtures, schedules, and loss weights; Section B.2 gives the training budgets and compute estimates. Experiment-specific settings, evaluation protocols, and results remain in their respective appendices.

B.1 Data Mixing

We specify mixtures by example counts or token proportions, converting token proportions to example-sampling rates using mean corpus or response lengths (Section 4). Loss weights do not affect the number of examples processed.

Midtraining data during pretraining. The study in Section 5.1 retains 256 packed pretraining sequences per update and adds $k \in \{4, 8, 16, 24, 32, 64\}$ packed midtraining sequences, all of length 2048. Mixing takes place over the final 8147 pretraining updates. The midtraining token fraction is therefore $k/(256 + k)$, and the number of added tokens is $8147 \times k \times 2048$. Here k counts packed sequences, each of which can contain multiple midtraining examples. The full midtraining corpus contains 39760 reasoning traces and 17879758 tokens; prompt, target, and end-of-sequence tokens all contribute to the loss. Every mixture reuses this corpus while preserving the original pretraining stream.

Midtraining sequences per update, k	Midtraining token fraction $k/(256 + k)$	Added midtraining tokens
4	1.54%	66740224
8	3.03%	133480448
16	5.88%	266960896
24	8.57%	400441344
32	11.11%	533921792
64	20.00%	1067843584

Table 2: Midtraining exposure during the final 8147 pretraining updates in Section 5.1. All configurations retain the original pretraining stream and reuse the same reasoning corpus.

Midtraining data during RL. The unified runs in Section 5.1 add m midtraining examples drawn from the 9996 Level 4 traces to each RL batch. The main sweep uses $m \in \{2, 4, 8, 16, 32\}$ with all six pretraining mixtures in Table 2. Section B.2 gives the corresponding corpus passes.

Pretraining data during RL. The three-source variant in Section C.8 uses the $k = 32$ initialization and $m \in \{4, 8, 16, 32\}$ midtraining examples per RL update. It additionally includes four pretraining-text sequences per update, with lengths chosen to contribute approximately 10% as many tokens as the midtraining data.

Token proportions in the real-world data study. In Section 5.2, we specify a target midtraining token proportion. Writing T_{mid} for midtraining tokens and T_{rl} for all RL response tokens, including unsuccessful responses, the corresponding fraction is

$$f = \frac{T_{\text{mid}}}{T_{\text{mid}} + T_{\text{rl}}}.$$

The percentages reported in the figures are configured targets for this fraction. We estimate response lengths before training to convert each target into a sampling probability for midtraining examples. This probability remains fixed for constant mixing and follows the chosen schedule otherwise, so realized token proportions can change as response lengths change.

For training progress $u \in [0, 1]$ and $a = 0.005$, the four schedules in Figure 7 are

$$f_{\text{constant}}(u) = a, \qquad f_{\text{full}}(u) = 2a(1 - u),$$
$$f_{\text{growth}}(u) = 2au, \qquad f_{\text{half}}(u) = 4a \max\{1 - 2u, 0\}.$$

Each has an average configured proportion a over training. Full and half annealing reach zero at the end and midpoint, respectively; the additional three-quarter-annealing schedules reach zero at three

quarters of training. These equal configured averages do not ensure equal realized token proportions. Section D.3 reports the schedule comparisons.

Loss weighting and normalization. The comparison in Section C.9 holds the initialization and example counts fixed and varies how the source losses are combined. Let N_R be the summed advantage-weighted response negative log-likelihood for RL, N_D the summed response negative log-likelihood on midtraining data, and T_R, T_D their respective response-token counts. Using one optimizer, the two objectives are

$$\mathcal{L}_{\text{unified}} = \frac{N_R + wN_D}{T_R + T_D}, \qquad \mathcal{L}_{\text{separate}} = \frac{N_R}{T_R} + \alpha\frac{N_D}{T_D}.$$

Here w weights midtraining tokens in the mixed loss, while α weights the separately normalized supervised loss. Relative to the two source means, unified training has supervised-loss coefficient wT_D/T_R and overall scale $T_R/(T_R + T_D)$. Its relative coefficient follows batch composition; α is fixed within each separate-loss run.

B.2 Compute Calculations

We report training exposure in processed examples and requested rollouts, alongside the model-FLOP estimates used in Figs. 1, 4 and 11. Repeated presentations count toward training exposure, even when the underlying corpus is fixed.

Pretraining, midtraining, and RL budgets. The 4000-update RL runs in Section 5.1 share a budget of 2048000 requested completions. Unified training additionally processes $4000m$ midtraining examples, or $4000m/9996$ passes through the Level 4 corpus. For $m = 2, 4, 8, 16, 32$, these are approximately 0.8, 1.6, 3.2, 6.4, and 12.8 passes; the separate-loss comparison also uses $m = 128$, corresponding to 51.2 passes. Increasing either the pretraining mixture k or the midtraining count m during RL adds compute by reusing the fixed corpus. Pretraining replay adds text tokens at unchanged RL-update and midtraining-data budgets.

Model-FLOP estimates. For Figs. 1, 4 and 11, we follow the dense-model convention of Shen et al. (2026), counting approximately $6NT$ FLOPs for training and $2NT$ for generation. Here $N = 123587328$ is the parameter count and T counts model-input tokens. Let T_{pre} denote the shared pretraining tokens, T_{mid} the additional midtraining tokens processed before RL, T_{rollout} the cumulative rollout tokens, and T_{mix} the midtraining tokens mixed into RL. The compute estimate is

$$C_{\text{total}} = 6N(T_{\text{pre}} + T_{\text{mid}}) + 8NT_{\text{rollout}} + 6NT_{\text{mix}}.$$

The rollout term includes generation and one policy-training pass; each sampled completion contributes its prompt and response tokens. Static-data counts likewise include all nonpadding input tokens, including positions masked from the loss. These compute counts therefore differ from the response-token counts used for RL mixing and loss normalization in Section B.1. The estimate omits attention-length corrections, padding overhead, activation recomputation, evaluation, and work discarded after restarts.

All runs retain the same 20B-token pretraining stream. Before RL, unified initializations add approximately 67M–1.07B midtraining tokens, while the selected staged reference processes 572M tokens over 32 epochs. The expanded staged comparison in Figure 11 uses 4–64 epochs, with 17879758 midtraining tokens per epoch. During RL, we accumulate recorded rollout lengths and reconstruct midtraining-token counts from the saved data-sampler states.

Each column of Figure 4 is capped at the final staged checkpoint's compute budget. The black frontier connects nondominated observed checkpoints from the selected unified and staged runs, with connecting segments as visual guides. It holds the best observed accuracy constant up to the budget limit without adding an evaluation point. The right panel of Figure 1 uses the same Level 6 evaluations and compute cap, showing the best observed accuracy within each budget separately for unified training and the staged reference. These step curves increase only when an evaluated checkpoint improves on the previous best accuracy. Figure 11 instead includes five staged budgets and caps each column at the largest staged endpoint cost. Its second row subtracts the shared pretraining cost, $6NT_{\text{pre}}$, from every checkpoint; this changes the horizontal spacing without changing frontier membership.

Real-world data experiments. The comparisons in Section 5.2 are not matched in total compute: the data mixture changes the work per update, and the asynchronous trainer can reuse responses. Section D.6 reports measured update throughput with the hardware and staleness caveats. These timings measure optimization throughput rather than model FLOPs or fresh-rollout generation.

C PRETRAINING, MIDTRAINING, AND RL: SETUP AND ADDITIONAL RESULTS

This appendix gives the experimental protocol and additional comparisons for Section 5.1.

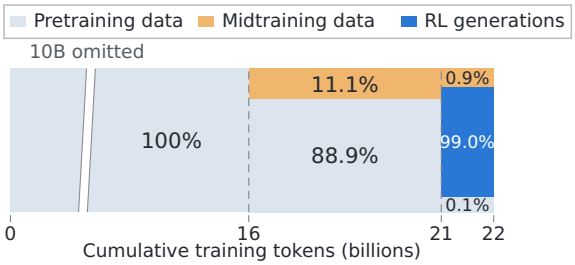


Figure 9: **Three-source data mixing.** An additional variant retains pretraining text during RL (Section C.8), with 534M midtraining tokens before RL and 3.2 passes during RL. Band heights are schematic; the break omits 10B pretraining tokens.

C.1 TASKS AND MIDTRAINING DATA

Number-assignment puzzles. Each puzzle asks the model to assign n sampled distinct integers bijectively to n people using natural-language clues. Starting from a known assignment, generation adds true clues until exhaustive permutation enumeration establishes uniqueness, then adds redundant clues. Clues express sums, differences, comparisons, parity, multiplication, and extrema. A single completion contains a boxed assignment, with binary reward for correctness of the full assignment. Table 3 defines Difficulty Levels 1–6 by jointly varying the number of people, the integer range, and the number of redundant clues. The experiments in Section 5.1 use Levels 4–6 for RL and evaluation.

Difficulty level	People	Integer range	Redundant clues	Midtraining questions	Reasoning traces
1	5	1–30	0	2476	9904
2	6	1–30	0	2473	9892
3	7	1–30	2	2492	9968
4	8	1–36	3	2499	9996
5	9	1–40	4	–	–
6	10	1–44	5	–	–

Table 3: Difficulty levels and midtraining-data counts. Redundant clues are added after the assignment is uniquely determined. Midtraining data covers Levels 1–4 with four traces per question; RL experiments evaluate Levels 4–6.

Oracle reasoning traces. The midtraining corpus contains 39760 traces for 9940 distinct questions across Levels 1–4. Questions containing multiplication, largest, or smallest clues are excluded from this corpus. A symbolic oracle propagates candidate sets and fixed assignments; changes in constraint order and verbalization produce alternative traces. We retain only complete traces with correct final assignments and remove duplicate trace text, yielding four traces per question. Final pretraining mixtures and staged supervised training use the full corpus. During RL, unified training reuses midtraining data only from Level 4: 9996 traces for 2499 questions.

Training and test separation. The Level 4 and Level 5 RL pools each contain 20000 questions, and the Level 6 pool contains 5000 questions, with 1024 held-out test questions per level. RL and test questions use the full clue vocabulary. Initialization evaluations on Levels 5–6 measure transfer beyond the difficulty levels covered by midtraining data; RL runs on these levels train on the corresponding difficulty directly.

C.2 Model and Pretraining

Model and original pretraining. Table 4 summarizes the 124M model and its original pretraining configuration. We train from scratch on a mixture of 75% ClimbMix (Diao et al., 2026)[2] and 25% Dolmino (Olmo et al., 2026),[3] consuming approximately 15B and 5B tokens, respectively. The original run processes 20000014336 tokens in total.

Setting	Value
Parameters / layers	123587328 / 12
Hidden / feedforward size	768 / 2048
Attention heads	12
Embeddings and normalization	Tied embeddings, rotary position embeddings, RMSNorm
Tokenizer / context length	GPT-2 / 2048 tokens
Global batch size	256 packed sequences of 2048 tokens
Training updates	38147
Optimizer / betas	AdamW / $(0.9, 0.95)$
Peak / minimum learning rate	6×10^{-4} / 6×10^{-5}
Weight decay / gradient clip	0.1 / 1.0
Learning-rate schedule	200-update warmup; constant through update 30518; cosine decay over the final 7629 updates

Table 4: **Model and original pretraining hyperparameters.** The learning-rate schedule is for the original 38147-update run; mixed-pretraining branches resume at update 30000 and adjust the schedule to preserve cooldown.

Mixed-pretraining initializations. Unified initializations branch from the shared checkpoint at update 30000 and train for a further 8147 updates. They preserve the original pretraining stream and optimizer state, with the schedule adjusted to preserve cooldown when batch size changes. Section B.1 gives the mixture construction and token counts in Table 2.

C.3 Staged Initialization and RL Training

Staged initialization. Staged training completes the original 20B-token pretraining run, then trains on all 39760 reasoning traces before RL. Table 5 gives the selected midtraining configuration and the shared RL settings. Here, midtraining uses prompt–response examples, with the same masking convention as SFT. Prompt tokens are masked, while target and end-of-sequence tokens contribute to the loss. The resulting initialization is followed by RL alone. Section C.6 reports the midtraining sweep. Figure 10 evaluates the same 32-epoch initialization before RL.

RL training. Both approaches use MaxRL with the RL settings in Table 5. Each rollout batch receives one optimization pass as a single global minibatch.

Unified training. Unified runs start from the mixed-pretraining initializations and include Level 4 midtraining data during RL using the mixtures in Section B.1. The main sweep uses all six initializations; Section C.8 examines the addition of pretraining data during RL. Section B.2 gives the common rollout budget and the conversion from midtraining examples per update to corpus passes.

Combining midtraining and RL losses. The alternative with separately normalized RL and supervised losses uses the same mixed-pretraining initialization and midtraining batch size as unified training. We compare the $k \in \{24, 32\}$ initializations, corresponding to approximately 400M and 534M midtraining tokens during final pretraining. For each initialization and difficulty level, we compare supervised weights $\alpha \in \{0.1, 0.5, 1\}$ at $m \in \{8, 32, 128\}$, with unified midtraining weight fixed at $w = 1$. Both methods use MaxRL for the RL component. Section B.1 defines the two loss functions and their normalization; Section C.9 presents the full comparison.

[2] https://huggingface.co/datasets/karpathy/climbmix-400b-shuffle
[3] https://huggingface.co/datasets/allenai/dolma3_dolmino_mix-100B-1025

Setting	Staged midtraining	RL (both approaches)
Optimizer	AdamW	AdamW
AdamW betas	$(0.9, 0.95)$	$(0.9, 0.999)$
Learning rate	5×10^{-5} peak	10^{-5}
Learning-rate schedule	5% linear warmup, then cosine decay	Constant, no warmup
Weight decay	0.01	0.01
Gradient clip	1.0	1.0
Training updates	39776 (32 epochs)	4000
Batch size	32 reasoning traces	32 prompts, 16 responses each (512 rollouts)
Maximum prompt length	—	512 tokens
Maximum response length	—	1280 tokens
Sampling temperature / top-p	—	1.0 / 1.0
Top-k sampling	—	Disabled
KL penalty / entropy bonus	—	None / none

Table 5: **Staged midtraining and shared RL hyperparameters.** The midtraining column specifies the 32-epoch staged reference in Figure 4; the RL column applies to both staged and unified training. Unified training adds the midtraining examples and loss weights specified in Section B.1; Section B.2 gives the exposure and compute calculations.

C.4 Evaluation Protocol

Accuracy and coverage. Evaluation uses 1024 held-out questions per evaluated level. Mixed-pretraining initialization evaluations use 16 samples per question, including the unified curves in Figure 10 and the step-zero points in the comparisons of loss normalization. The staged reference in Figure 10, the midtraining sweep in Section C.6, and evaluations during and after RL use 64 samples per question. We sample at temperature 1.0 and top-p 1.0, with generation seed zero and a maximum of 1280 response tokens. Accuracy denotes sampled pass@1, and coverage uses the unbiased pass@k estimator of Chen et al. (2021), averaged across questions. For c correct completions among n samples, the per-question estimator is

$$\widehat{\text{pass@}k} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}, \qquad 1 \le k \le n,$$

where $\binom{n-c}{k} = 0$ for $n - c < k$. Coverage measures whether the samples contain a correct answer under the grader; it does not assume that the model can select that answer itself. Greedy accuracy is treated separately from sampled pass@1.

Pretraining retention. Figure 5 evaluates the initializations before RL. Held-out language-model loss uses about 400000 fixed tokens from each of ClimbMix and Dolmino in 1024-token windows, combined with weights 0.75 and 0.25. The general-capability score averages MMLU (Hendrycks et al., 2021a) accuracy, ARC-Easy (Clark et al., 2018) normalized accuracy, HellaSwag (Zellers et al., 2019) normalized accuracy, TruthfulQA MC2 (Lin et al., 2022), and WinoGrande (Sakaguchi et al., 2020) accuracy.

C.5 Reasoning Performance before RL

Figure 10 evaluates the models after mixed pretraining, before RL, across all six midtraining exposures. The staged reference completes pretraining and then uses 32 epochs of midtraining at a peak learning rate of 5×10^{-5}, matching the initialization in Figure 4. Both approaches use the same midtraining corpus, but their midtraining exposure and total computation differ; this comparison characterizes the initializations rather than matching their training budgets.

On Level 4, increasing midtraining exposure consistently improves both accuracy and coverage. The largest mixture exceeds the staged reference at higher sampling budgets on Level 4, despite lower pass@1; staged training retains higher accuracy and coverage on Level 5. Transfer beyond the difficulty levels covered by midtraining data remains limited for both approaches: performance is substantially lower on Level 5 and close to zero on Level 6. Together with the retention results in Figure 5, these evaluations show that mixed pretraining develops task-specific ability, but does not by itself yield transfer to harder problems. The subsequent RL experiments train directly on the evaluated difficulty level and therefore measure adaptation to those levels.

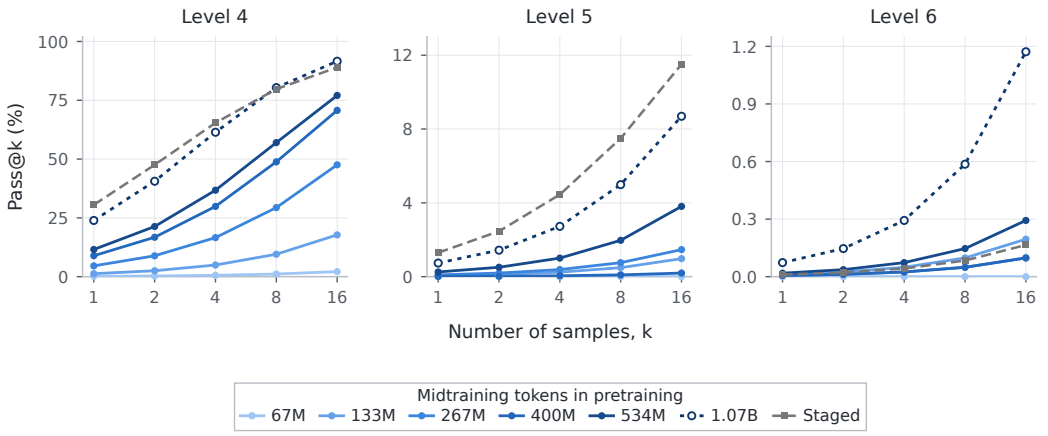


Figure 10: **Reasoning accuracy and coverage before RL.** Colors distinguish midtraining exposure during final pretraining; dashed gray curves show the 32-epoch staged initialization. All models use midtraining data from Levels 1–4, so Levels 5–6 test transfer to harder problems. Pass@k uses 16 samples per question for unified initializations and 64 for the staged reference, restricted to the common range $k \leq 16$. Each level contains 1024 held-out questions. Vertical scales differ across panels.

C.6 Staged Midtraining Sweep

We vary the number of midtraining epochs and peak learning rate on the same 39760 reasoning traces, keeping the base model and remaining training settings fixed. Table 6 reports accuracy and coverage immediately after midtraining, before RL, using 64 samples for each of 1024 held-out questions at each difficulty level. Level 4 is represented in the midtraining corpus, whereas Levels 5–6 test transfer to harder problems.

		Level 4			Level 5			Level 6		
Epochs	Peak LR	pass@1	pass@8	pass@32	pass@1	pass@8	pass@32	pass@1	pass@8	pass@32
4	2×10^{-4}	17.30	67.77	93.27	0.16	1.22	4.26	0.002	0.01	0.05
8	5×10^{-5}	19.02	71.45	94.76	0.15	1.13	3.77	0.003	0.02	0.10
8	2×10^{-4}	22.01	70.90	91.58	0.43	2.94	8.02	0.009	0.07	0.27
8	4×10^{-4}	21.76	69.82	89.87	0.02	0.18	0.59	0.002	0.01	0.05
16	5×10^{-5}	24.64	76.62	93.95	0.71	4.62	11.82	0.006	0.05	0.20
16	2×10^{-4}	28.27	76.94	92.34	0.29	2.05	5.87	0.008	0.06	0.22
32	5×10^{-5}	30.54	79.76	93.89	1.30	7.49	16.05	0.011	0.08	0.32
32	2×10^{-4}	36.85	85.61	95.79	1.52	8.03	15.88	0.020	0.15	0.57
64	5×10^{-5}	34.32	82.82	94.46	1.28	7.07	14.37	0.021	0.17	0.61
64	2×10^{-4}	41.13	86.98	95.62	1.05	5.72	12.21	0.009	0.07	0.23

Table 6: **Staged midtraining sweep before RL.** All configurations use the same base model and the full midtraining corpus. Values are percentages; pass@1 measures accuracy, and pass@8 and pass@32 measure coverage at larger sampling budgets. The shaded row is the initialization used for the staged reference in Figure 4.

Higher accuracy immediately after midtraining does not necessarily yield a better initialization for RL. At each epoch budget, we therefore select the peak learning rate with the highest mean pass@1 across Levels 4–6 after 4000 RL updates, among the settings in Table 6. The same learning rate is therefore used for all three levels at each budget. Table 7 reports the selected configurations; the 32-epoch setting also has the highest mean final accuracy across these budgets and serves as the main-text reference.

Epochs	Peak LR	Level 4	Level 5	Level 6	Mean
4	2×10^{-4}	68.91	0.00	4.65	24.52
8	2×10^{-4}	72.24	50.76	21.23	48.08
16	2×10^{-4}	73.94	54.71	19.58	49.41
32	5×10^{-5}	83.26	73.39	34.19	63.61
64	2×10^{-4}	75.05	64.59	24.46	54.70

Table 7: **Selected staged configurations after RL.** Pass@1 (%) after 4000 RL updates. Learning rates maximize mean final accuracy across the three difficulty levels at each midtraining budget. The shaded row is the main-text reference.

C.7 Accuracy and Compute across Midtraining Budgets

Figure 3 in the main text reports final accuracy across the unified mixtures. Here we extend the compute comparison across staged midtraining budgets.

Figure 11 extends Figure 4 to staged training with 4, 8, 16, 32, and 64 midtraining epochs, using the learning rates selected in Table 7. It includes the full trajectories of these five staged configurations and the unchanged unified runs. Including the additional staged budgets shifts part of the observed frontier to staged training at lower compute, while unified training achieves the highest accuracy within the displayed budget on all three levels. The comparison therefore supports the accuracy–compute advantage of unified training beyond a single staged initialization.

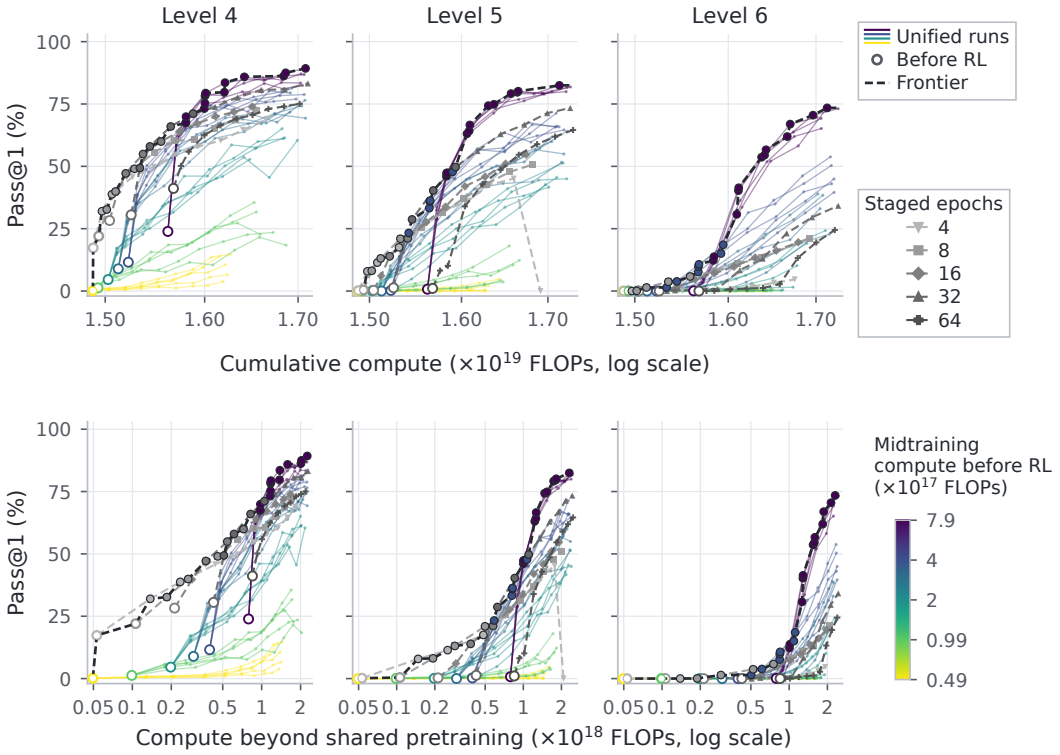


Figure 11: **Accuracy–compute frontier across midtraining budgets.** Colored curves show unified runs, and gray dashed curves show staged training with the selected learning rate at each epoch budget. Open circles mark performance before RL. The black frontier is recomputed from all observed checkpoints shown, with colored and gray markers identifying unified and staged checkpoints. Both rows show the same evaluations: the top includes all training compute, while the bottom subtracts the shared pretraining cost. Each column ends at the largest final compute budget among its staged runs; compute is calculated as in Section B.2.

C.8 Retaining Pretraining Data during RL

We compare unified training with and without pretraining data during RL, using the same initialization and midtraining batch size in each pair. All runs start from the $k = 32$ mixed-pretraining model, which processes approximately 534M reasoning tokens during final pretraining, and use $m \in \{4, 8, 16, 32\}$ midtraining examples per RL update. The pretraining-data mixture is defined in Section B.1. Both variants use 4000 RL updates and the same midtraining exposure; retaining pretraining data increases the total number of training tokens.

Figure 12 shows that the effect of retaining pretraining data on accuracy varies with difficulty and midtraining reuse. At the largest midtraining batch, adding pretraining data improves accuracy on Levels 5–6 but slightly reduces it on Level 4; across the full sweep, it provides no consistent accuracy advantage.

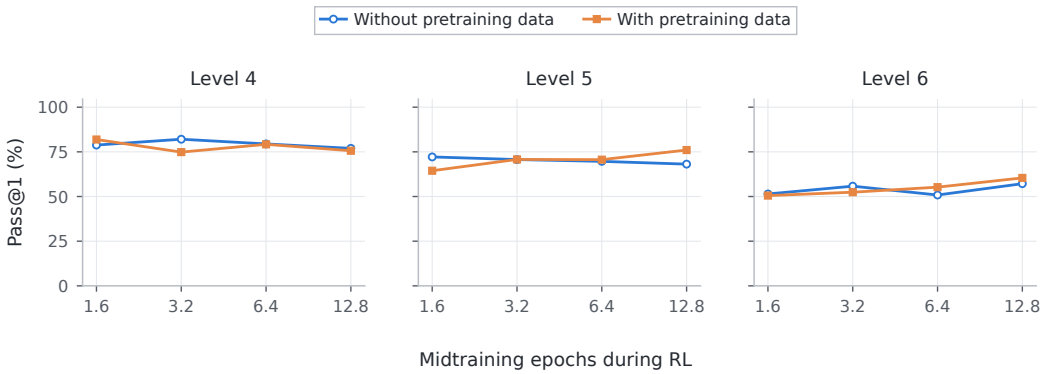


Figure 12: **Effect of retaining pretraining data during RL.** Both variants use unified training with the same mixed-pretraining initialization and midtraining batch size at each point. The added pretraining data contributes approximately 10% as many tokens as the midtraining data. Each column trains and evaluates at the indicated difficulty level; pass@1 is measured after 4000 RL updates using 64 samples per question.

C.9 Combining Midtraining and RL Losses

Figure 13, Figure 14, and Figure 15 compare unified training with adding a separately normalized supervised loss to RL on Difficulty Levels 4–6. Each figure includes the two mixed-pretraining initializations, trained with approximately 400M and 534M midtraining tokens. Within each comparison, both methods use the same initialization, midtraining corpus, midtraining batch size, and RL rollout budget. The objectives are defined in Section B.1, the compared settings in Section C.3, and the exposure conversion in Section B.2.

The comparison tests whether weighting the source losses through the mixed batch is effective relative to choosing a fixed coefficient between separately normalized losses. Larger supervised-loss coefficients often slow improvement and reduce final accuracy, particularly on the harder levels. With unit midtraining weight throughout, unified training achieves higher final accuracy than the alternative at every tested coefficient on Level 4 for both initializations. On Levels 5–6, a tuned supervised-loss coefficient can perform better, and the relative ranking depends on the initialization and midtraining batch size. Thus, the shared token-mean objective provides a competitive way to determine the relative source weights from the data mixture.

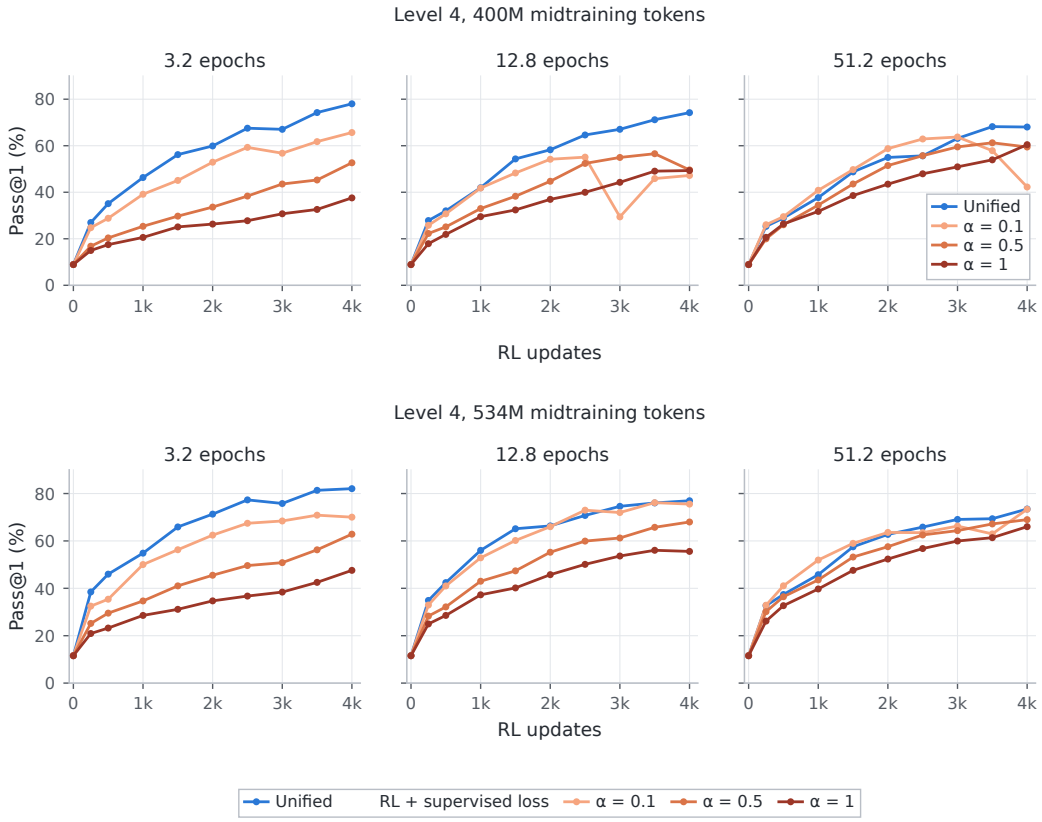


Figure 13: **Combining midtraining and RL losses on Difficulty Level 4.** The top and bottom rows use initializations trained with 400M and 534M midtraining tokens, respectively. Columns vary midtraining exposure during RL; curves show pass@1 through 4000 RL updates. Unified training uses $w = 1$, while the alternative varies the coefficient α on its separately normalized supervised loss.

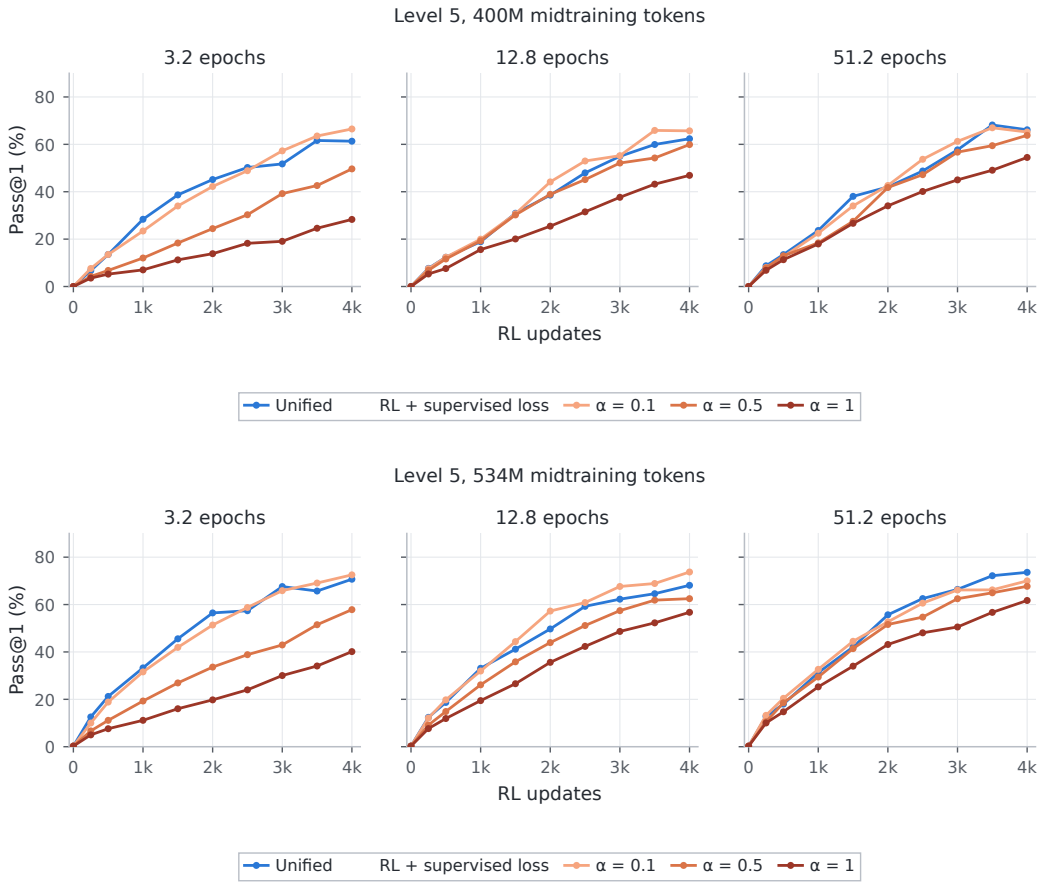


Figure 14: **Combining midtraining and RL losses on Difficulty Level 5.** The top and bottom rows use initializations trained with 400M and 534M midtraining tokens, respectively. Columns vary midtraining exposure during RL, with the same objectives and loss weights as Figure 13.

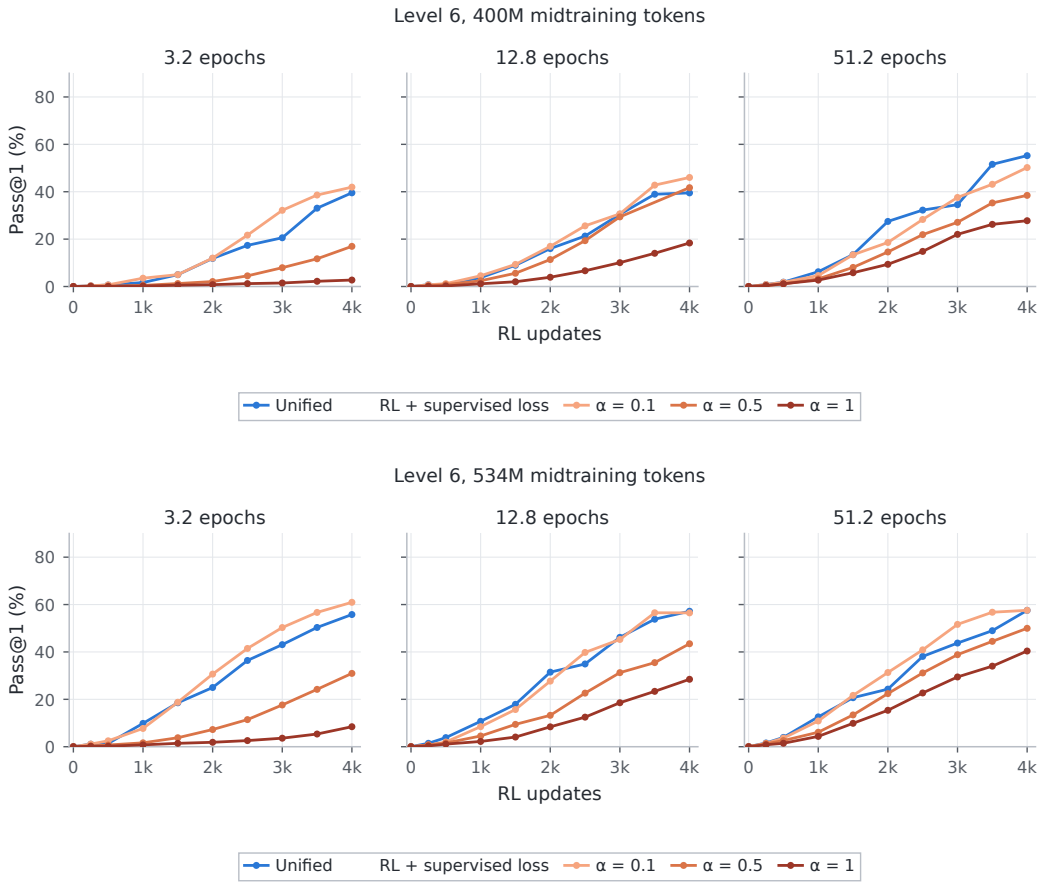


Figure 15: **Combining midtraining and RL losses on Difficulty Level 6.** The top and bottom rows use initializations trained with 400M and 534M midtraining tokens, respectively. Columns vary midtraining exposure during RL, with the same objectives and loss weights as Figure 13.

D Details of Training with Real-world Midtraining Data

This appendix gives the setup and additional results for Section 5.2. Endpoint comparisons use final checkpoints, without selecting a different checkpoint for each metric.

D.1 Experimental Setup

Model and data. We train the model from scratch on 354.53B ClimbMix pretraining tokens (Diao et al., 2026), followed by 17.73B OLMo-3 Dolmino midtraining tokens (Olmo et al., 2026). Midtraining also includes one epoch of supervised fine-tuning on packed verified examples excluding GSM8K. The model has 756819456 parameters, 24 layers, hidden width 1536, 12 attention and key/value heads, and context length 2048. This yields the shared base model; staged training additionally uses the corpus described below before RL. Math12K supplies 11977 RL prompts after length filtering; reward is binary correctness of the last boxed answer under fixed answer-normalization rules. The additional corpus contains 240M tokens from 24 midtraining components, including web text, mathematics, code, question answering, and reasoning traces. Training on this corpus uses document continuation with unit token weights, except for the code-data replacement described in Section D.4. Table 8 gives the pretraining and Dolmino midtraining settings.

Setting	Pretraining	Midtraining
Optimizer	AdamW	
Adam (β_1, β_2); ε	$(0.9, 0.95)$; 10^{-8}	
Weight decay; gradient clipping norm	0.1; 1	
Global / per-device microbatch (sequences)	2048 / 4	
Sequence length (tokens)	2048	
Peak / minimum learning rate	4×10^{-4} / 4×10^{-5}	
Learning-rate schedule	Warmup, constant, cosine decay	
Warmup updates	845	177
Final decay updates	16905	845
Total updates	84527	4226

Table 8: **Pretraining and midtraining hyperparameters for the 0.76B base model.** The midtraining column describes training on the Dolmino corpus.

Data mixing. Section B.1 defines the configured token proportions, sampling probabilities, and schedules used here; Section B.2 describes their budget conventions.

Training. Unified training without midtraining data provides the RL-only baseline. Staged training starts from the shared base model, trains on the additional 240M-token corpus during midtraining, and then uses on-policy MaxRL with baseline subtraction. It therefore has access to the same additional static corpus as unified training, but uses it before RL. We do not see significant differences with or without the additional midtraining data for staged training.

Unified training publishes weights every four updates and retains the newest generation round from each generator, even when it is older than four updates. The optimizer, batch, and rollout settings are listed in Table 9.

Evaluation. Math averages MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2024) and GSM8K (Cobbe et al., 2021); Coding averages the EvalPlus HumanEval+ and MBPP+ suites (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023).[4] The two benchmarks receive equal weight in each metric. Evaluation covers all 500 MATH-500, 1319 GSM8K, 164 HumanEval+, and 378 MBPP+ problems. Pass@1 uses 16 responses per problem. Math pass@32, pass@64, and pass@256 use a separate 256-response evaluation; Coding pass@32 uses a separate 32-response evaluation. We use the pass@k estimator in Section C.4 (Chen et al., 2021). Coverage measures whether sampling produces a correct answer, without assuming the model can select it. Sampled evaluations use temperature 1, top-k 50, top-p 0.95, and context length 2048, with generation caps of 1024 tokens for math and 768 for coding. SciQ (Welbl et al., 2017) and ARC-Challenge (Clark et al., 2018) use

[4] MBPP+ was released through EvalPlus: https://github.com/evalplus/evalplus/releases/tag/v0.2.0.

Setting	Unified	Staged RL
Optimizer	AdamW	AdamW
Adam (β_1, β_2)	$(0.9, 0.999)$	$(0.9, 0.999)$
Weight decay	0	0
Learning rate	10^{-6}	10^{-6}
Learning-rate schedule	Constant, no warmup	Constant, no warmup
Configured global batch (sequences)	512	512
Microbatch per device (sequences)	4	4
Responses per prompt	16	16
RL data collection	Asynchronous generators	32 fresh prompts per update
Prompt + response limit (tokens)	2048	2048
Maximum response length (tokens)	1024	1024
Temperature / top-k / top-p	1 / 50 / 0.95	1 / 50 / 0.95
Reward-baseline subtraction	No	Yes

Table 9: **Mathematical RL hyperparameters.**

greedy decoding with visible answer choices, an eight-token cap, and strict leading-label scoring. We also report MMLU (Hendrycks et al., 2021a).

Held-out loss. We measure token loss separately on 24 midtraining components and average the component losses equally, using 3608122 scored tokens in total. Loss change is measured relative to the base model, whose loss is 1.4352 nats/token. Negative change means lower loss than the base model. Evaluation documents were excluded from the current midtraining corpus and checked against an earlier 10B-token training inventory; this does not establish exclusion from the full midtraining history.

Base-model performance. Table 10 reports the base model's performance before mathematical RL, using the evaluation protocol above.

Benchmark	Pass@1	Pass@32
MATH-500	6.54	43.76
GSM8K	17.84	71.63
Math (average)	12.19	57.69
HumanEval+	11.97	32.32
MBPP+	21.05	57.14
Coding (average)	16.51	44.73

Metric	Value
MMLU accuracy	37.00
SciQ accuracy	51.00
ARC-Challenge accuracy	31.23
Held-out loss	1.4352

Table 10: **Base-model performance before mathematical RL.** Scores are percentages, except held-out loss in nats/token. Pass@1 uses 16 responses per problem; pass@32 uses 256 responses for math and 32 for coding. Math and Coding are the equally weighted benchmark averages defined in Section D.1.

D.2 Effect of the Midtraining Data Proportion

We evaluate proportions of 0, 0.1, 0.2, 0.5, 1, 2, and 5%. Figure 16 shows training curves for six selected proportions, including 2% and 5%; the full sweep also includes 0.2%. Curves connect evaluations during training without smoothing.

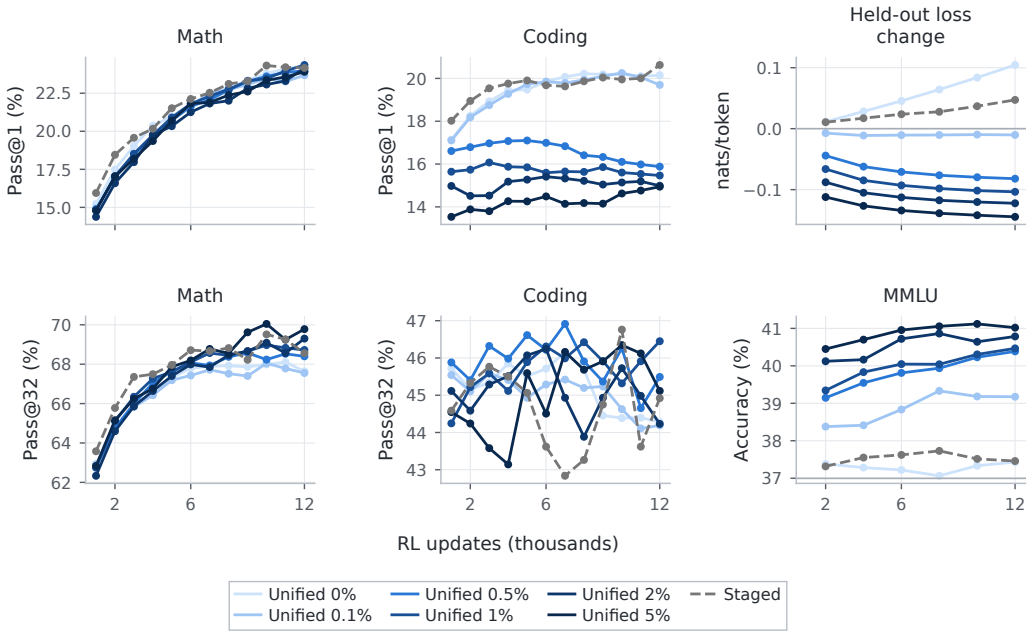


Figure 16: **Training curves for different midtraining data proportions.** Constant mixing at 0, 0.1, 0.5, 1, 2, and 5%, with the staged reference dashed. Math and Coding use the benchmark averages and evaluation sample counts in Section D.1.

Mixing	MATH pass@1	GSM pass@1	MATH pass@64	GSM pass@64	MATH pass@256	GSM pass@256
0%	16.17	32.46	63.97	84.53	75.60	92.04
0.1%	15.64	31.72	64.80	83.99	77.60	92.04
0.2%	16.38	31.86	65.61	83.59	75.80	91.81
0.5%	16.13	31.82	65.34	84.62	76.20	92.29
1%	16.85	31.86	66.28	84.58	77.40	92.19
2%	16.51	31.80	66.26	85.22	76.60	92.57
5%	16.33	31.49	67.33	85.36	78.00	92.87
Base model	6.54	17.84	53.67	80.15	70.20	91.51
Staged	16.98	31.35	66.17	84.50	78.00	92.34

Table 11: **Mathematical accuracy and coverage at different midtraining proportions.** Scores (%) before RL (Base model) and after RL. MATH and GSM denote MATH-500 and GSM8K. Pass@1 uses 16 responses; pass@64 and pass@256 use 256.

Mixing	Δloss	MMLU	SciQ	ARC-C	MBPP+	HumanEval+
0%	+0.1045	37.43	54.73	30.60	25.03	15.27
0.1%	−0.0102	39.18	57.70	31.66	24.95	14.44
0.2%	−0.0478	40.05	59.30	31.83	22.54	13.83
0.5%	−0.0821	40.39	62.07	32.94	20.19	11.57
1%	−0.1036	40.47	63.47	31.94	19.51	11.42
2%	−0.1224	40.78	66.40	31.14	19.00	10.98
5%	−0.1446	41.02	66.60	32.94	19.06	10.82
Base model	0.0000	37.00	51.00	31.23	21.05	11.97
Staged	+0.0472	37.46	53.00	31.48	25.74	15.51

Table 12: **Held-out loss and performance beyond math.** Scores are percentages; Δloss is relative to the base model before RL, in nats/token. Coding columns report pass@1 from 16 responses. ARC-C denotes ARC-Challenge.

Individual benchmarks. Figure 17 and Figure 18 show the benchmarks underlying the main averages. Larger midtraining proportions continue to lower held-out loss, while coding accuracy remains below the RL-only baseline. Coding coverage can nevertheless improve: at 5%, MBPP+ pass@16 is 51.59%, compared with 48.24% for the RL-only baseline. At proportions of 0.5% and above, coding accuracy also falls below the base model (Table 10).

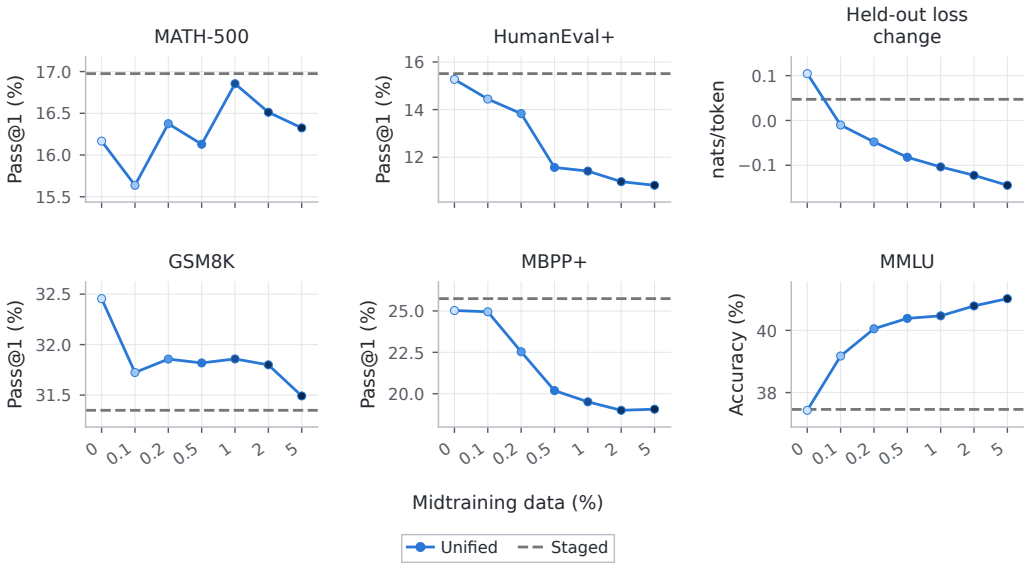


Figure 17: **Accuracy on individual benchmarks.** MATH-500, GSM8K, HumanEval+, and MBPP+ pass@1 use 16 responses per problem. The staged reference is dashed.

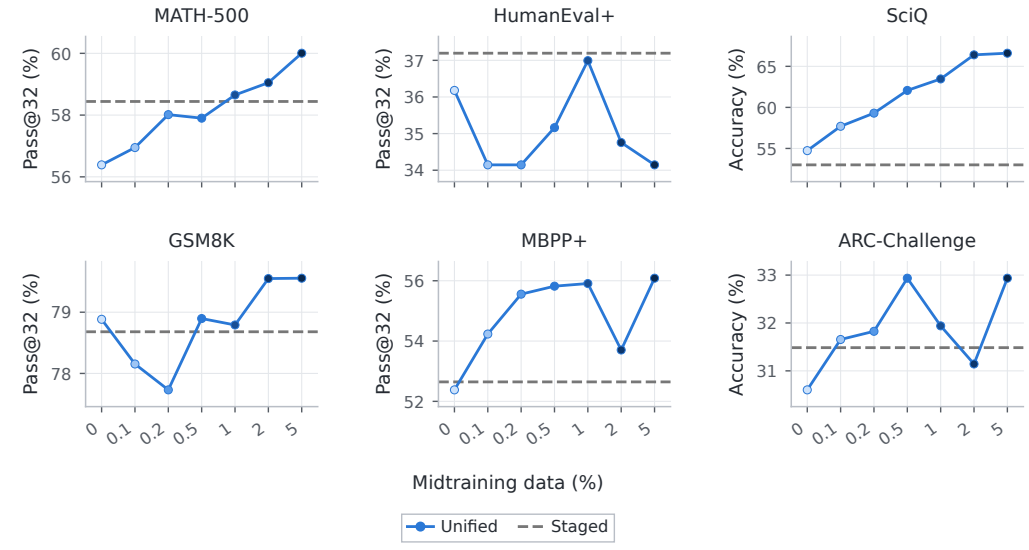


Figure 18: **Coverage and knowledge accuracy across midtraining proportions.** Math pass@32 uses 256 responses per problem and coding pass@32 uses 32. SciQ and ARC-Challenge use greedy evaluation. All seven proportions are included.

D.3 Data Mixing Schedules

Schedules with the same average proportion. The four schedules have an average configured midtraining proportion of 0.5%; their definitions and token-counting conventions are collected in Section B.1.

Figure 19 and Table 13 give the full results. After midtraining data is removed halfway through training, held-out loss increases while coding accuracy improves and SciQ gains persist. Across the four schedules, Math pass@1 spans 23.38–24.51%, Math pass@32 spans 67.67–68.58%, and MMLU spans 39.71–40.65%. Coding pass@32 varies from 44.89% to 47.53%, with a different ordering from pass@1, and ARC-Challenge also varies.

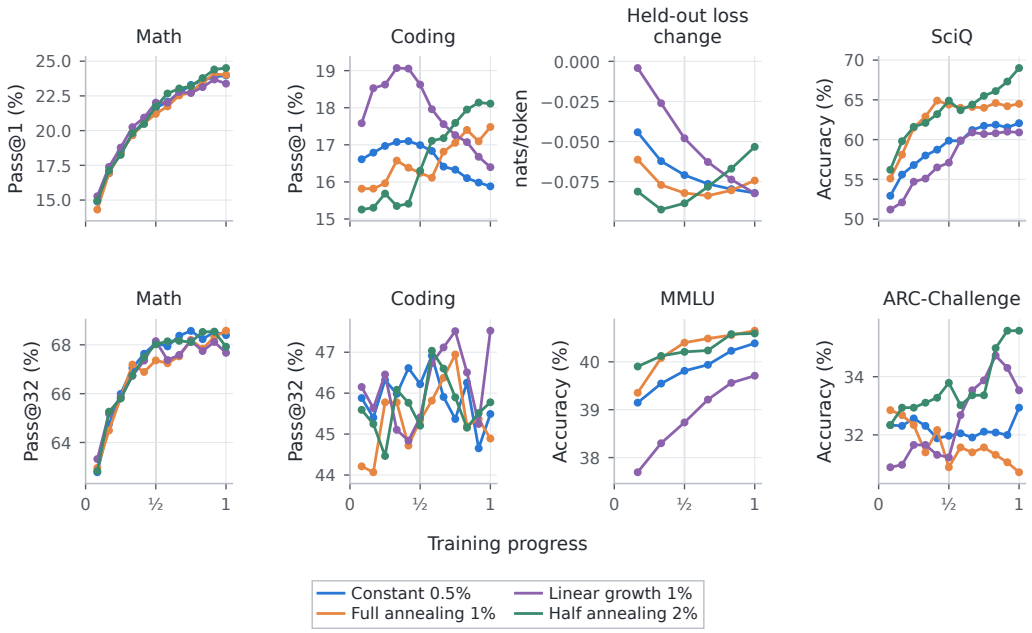


Figure 19: **Full results for the four mixing schedules.** The same runs as Figure 7, including Math and Coding accuracy and coverage, held-out loss change, MMLU, SciQ, and ARC-Challenge. The vertical line marks the midpoint of training.

Schedule	Math pass@1	Math pass@32	Coding pass@1	Coding pass@32	Δloss	MMLU
Constant 0.5%	23.97	68.40	15.88	45.49	−0.0821	40.39
Full anneal 1%	24.02	68.58	17.48	44.89	−0.0743	40.65
Ramp to 1%	23.38	67.67	16.40	47.53	−0.0823	39.71
Half anneal 2%	24.51	67.92	18.11	45.78	−0.0533	40.59

Table 13: **Results for the four mixing schedules.** Each schedule has an average configured midtraining proportion of 0.5%. Scores are percentages, except Δloss in nats/token. SciQ and ARC-Challenge appear in Figure 19.

Additional schedules. Figure 20 compares nine schedules: constant 0.2 or 0.5%, full decay from 0.2 or 0.5%, half decay from 1, 2, or 5%, and three-quarter decay from 1 or 2%. These settings vary both the amount of midtraining data and when it is used, so they do not isolate the effect of schedule timing alone.

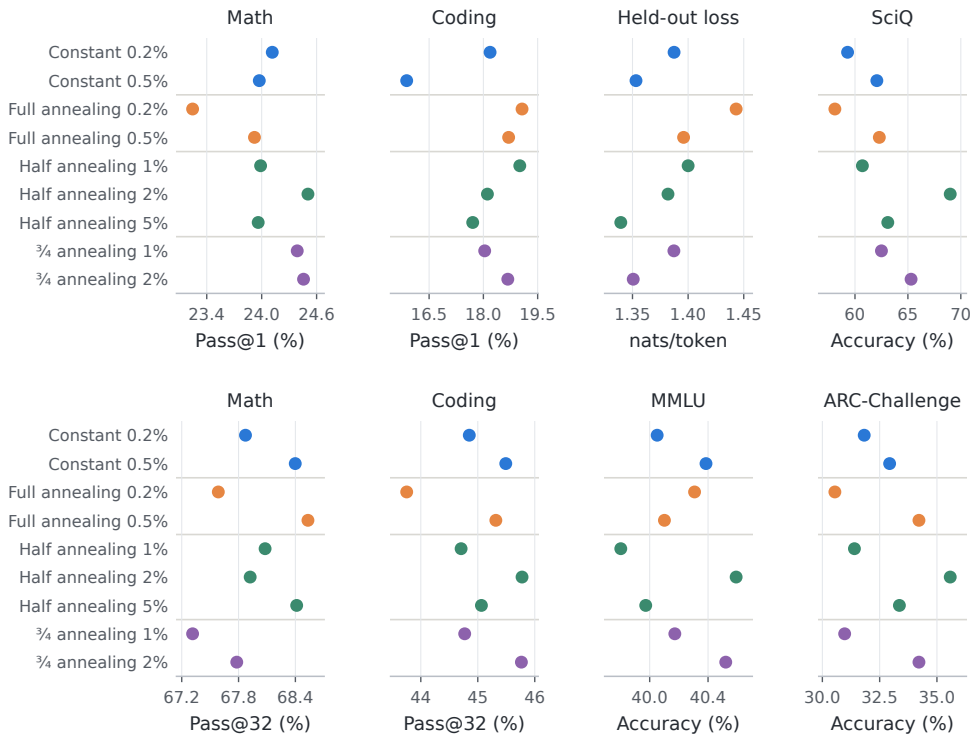


Figure 20: **Results for additional mixing schedules.** Full, half, and three-quarter annealing reach zero at the end, midpoint, and three-quarter point of training, respectively. Held-out loss is absolute.

D.4 Effect of Code Data

We examine whether the lower coding accuracy comes from the corpus's three designated code sources: CraneCode, StackEdu-FIM, and Code Meta-Reasoning, together accounting for 24.8% of its tokens. One experiment removes these sources and preserves the relative proportions of the remaining data. The other replaces their allocation with execution-tested Python answers from OpenCodeInstruct (Ahmad et al., 2025), using chat formatting and assistant-only loss. Removing the designated sources does not remove all possible code from the corpus, and replacement changes formatting and masking as well as content.

We compare the three corpora under constant 0.5% mixing and full decay from 0.5%. Figure 21 and Table 14 show only modest or inconsistent improvements in coding accuracy after removal or replacement.

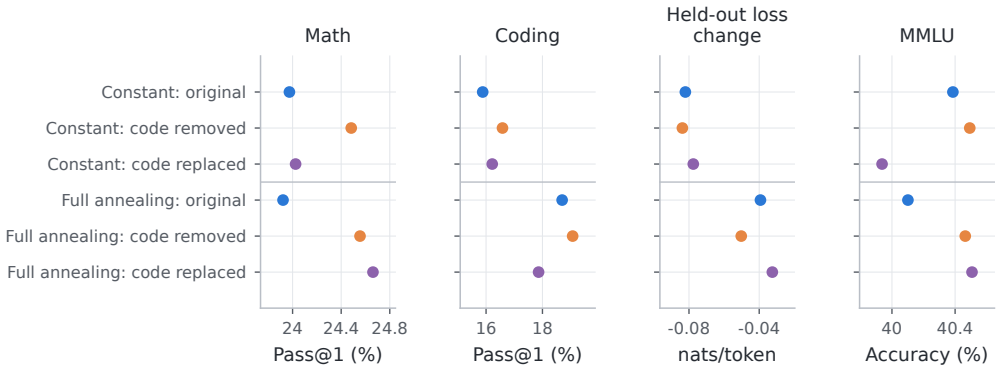


Figure 21: **Effect of removing or replacing code data.** The original corpus, the corpus with designated code sources removed, and the corpus with those sources replaced by verified coding examples. Each is evaluated under constant 0.5% mixing and full annealing from 0.5%. Math and Coding are the benchmark averages defined in Section D.1.

Setting	MATH	GSM	Δloss	SciQ	MBPP+	HumanEval+
constant: original	16.13	31.82	−0.0821	62.07	20.19	11.57
constant: code removed	16.41	32.55	−0.0838	60.00	20.82	12.35
constant: code replaced	16.11	31.94	−0.0775	63.40	20.97	11.47
anneal: original	15.69	32.16	−0.0393	62.30	24.06	13.34
anneal: code removed	16.78	32.34	−0.0502	59.90	24.04	14.10
anneal: code replaced	16.78	32.55	−0.0325	60.40	22.77	12.96

Table 14: **Results with different code data.** Constant 0.5% mixing and full decay from 0.5%, each with three corpus variants. Math and coding scores are pass@1 (%); SciQ reports accuracy (%). MATH denotes MATH-500, GSM denotes GSM8K, and loss change is in nats/token.

D.5 MaxRL versus REINFORCE

We compare MaxRL and REINFORCE under constant midtraining proportions of 0.5, 1, 2, and 5%, and under schedules that decrease linearly from 1 or 2% to zero halfway through training. The algorithms use the same base model and corpus, with results grouped by the configured mixing proportion and schedule. For a group of $N = 16$ responses with empirical success rate $\hat{p}$, MaxRL gives successful responses weight $1/\hat{p}$, whereas REINFORCE gives them unit weight; unsuccessful responses receive zero weight in both cases. Neither algorithm subtracts a reward baseline.

We do not rescale the RL weights to have the same mean across algorithms. Changing the algorithm therefore changes both the relative weighting of prompts and the contribution of RL to the loss relative to midtraining.

Constant 2% mixing. Figure 22 gives the training curves for the comparison in Table 15. The higher mathematical coverage of MaxRL and the lower held-out loss of REINFORCE persist through the later part of training.

Method	Math pass@1	Math pass@32	Coding pass@1	Coding pass@32	Δloss	MMLU	SciQ	ARC-C
MaxRL	24.16	69.30	14.99	44.23	−0.122	40.78	66.40	31.14
REINFORCE	24.05	67.19	15.67	46.21	−0.151	40.91	72.70	37.71

Table 15: **MaxRL and REINFORCE with a constant midtraining proportion of 2%.** Scores are percentages, except held-out loss change relative to the base model, measured in nats/token. ARC-C denotes ARC-Challenge. Training curves and comparisons across mixing settings appear in Figs. 22 and 23.



Figure 22: **MaxRL and REINFORCE with a constant midtraining proportion of 2%.** Training curves for the runs in Table 15, evaluated using the protocol in Section D.1. Both algorithms use unified training; blue curves use MaxRL and orange curves use REINFORCE.

Comparison across mixing settings. Across the six settings in Figure 23, MaxRL consistently achieves higher mathematical coverage, while REINFORCE yields lower held-out corpus loss and higher SciQ and ARC-Challenge accuracy. The effects on mathematical accuracy and coding depend on the mixing proportion and schedule.

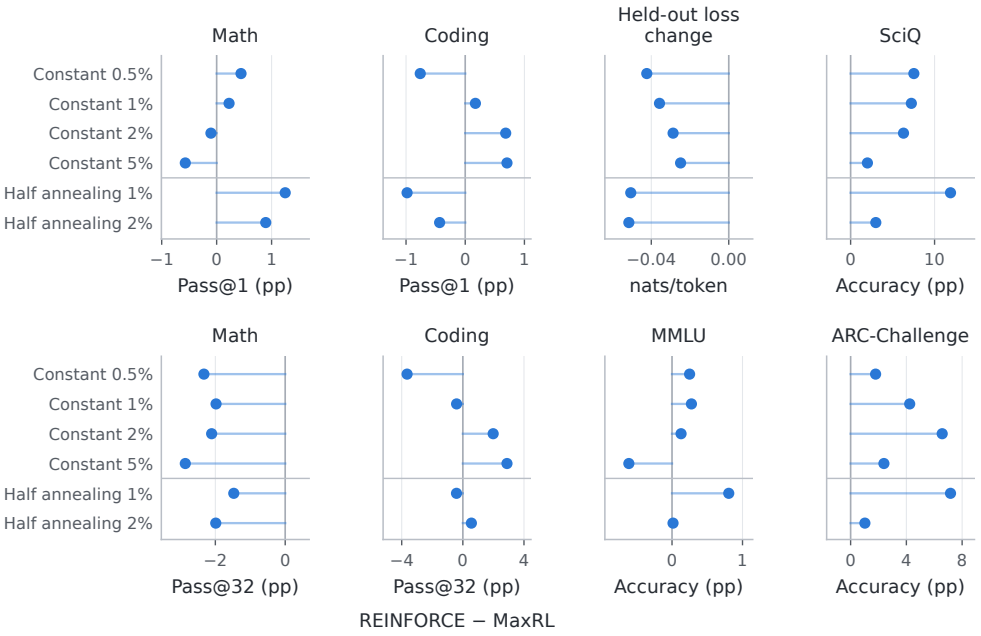


Figure 23: **Final MaxRL–REINFORCE comparison across mixing settings.** Differences are REINFORCE minus the corresponding MaxRL reference, in percentage points except loss in nats/token.

Table 16 reports the corresponding absolute results for both algorithms.

Mixing setting	Method	Math pass@1	Math pass@32	Coding pass@1	Coding pass@32	Δloss	MMLU	SciQ	ARC-C
Constant 0.5%	MaxRL	23.97	68.40	15.88	45.49	−0.082	40.39	62.07	32.94
	REINFORCE	24.42	66.07	15.12	41.84	−0.124	40.64	69.60	34.73
Constant 1%	MaxRL	24.36	68.72	15.47	46.45	−0.104	40.47	63.47	31.94
	REINFORCE	24.58	66.74	15.64	46.04	−0.139	40.74	70.70	36.18
Constant 2%	MaxRL	24.16	69.30	14.99	44.23	−0.122	40.78	66.40	31.14
	REINFORCE	24.05	67.19	15.67	46.21	−0.151	40.91	72.70	37.71
Constant 5%	MaxRL	23.91	69.78	14.94	45.12	−0.145	41.02	66.60	32.94
	REINFORCE	23.34	66.92	15.65	48.00	−0.170	40.41	68.60	35.32
Half anneal 1%	MaxRL	23.99	68.08	19.01	44.71	−0.035	39.80	60.70	31.40
	REINFORCE	25.24	66.61	18.02	44.29	−0.086	40.61	72.60	38.57
Half anneal 2%	MaxRL	24.51	67.92	18.11	45.78	−0.053	40.59	69.00	35.58
	REINFORCE	25.40	65.93	17.68	46.34	−0.105	40.61	72.00	36.60

Table 16: **MaxRL and REINFORCE across midtraining proportions and schedules.** Final results for the six comparisons in Figure 23. Half annealing decreases the stated initial proportion linearly to zero halfway through training. Scores are percentages, except held-out loss change relative to the base model in nats/token. ARC-C denotes ARC-Challenge.

D.6 Unified Training with Zero Mixing versus Staged RL

The 0% mixing condition uses the same asynchronous trainer and reward weighting as the other unified runs, providing a control for the effect of adding midtraining data. We compare this control with the staged RL reference to characterize the differences between the two implementations. Both start from the same base model and train exclusively on mathematical RL data. At zero mixing, the unified objective contains only the RL term, so the two approaches coincide when data generation and optimization are identical. Our implementations differ in asynchronous execution and reward-baseline subtraction, as detailed below.

Asynchronous execution and data staleness. The unified trainer consumes responses generated by earlier policies while generation continues in parallel, following Section 4. Staged RL instead collects responses from the current policy before optimization. Asynchronous execution removes this synchronization requirement, allowing the trainer to proceed with available responses at the cost of policy staleness. We publish updated weights every four optimizer updates, but this interval is not a bound on the age of consumed data: the buffer retains the newest round from each generator even when its generating policy is more than four updates old.

To quantify staleness in practice, we inspected the newest completed round from each of eight generators at two points during training with constant or full-decay 0.2% and 0.5% mixing, using the same publication interval and buffer recency settings. These rounds were generated by policies 12–44 optimizer updates behind the latest logged trainer step. These snapshots measure the age of available rollouts; the policy versions of the examples actually consumed by the trainer were not logged, so their age distribution cannot be recovered exactly.

Reward-baseline subtraction. Unified training weights responses by $r/\hat{p}$, whereas staged MaxRL uses $(r-\hat{p})/\hat{p}$, omitting numerical stabilizers here. Replacing $\hat{p}$ by the population pass rate $p_\theta(x) > 0$, the two weights yield the same expected on-policy gradient because

$$\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)] = 0.$$

Thus, subtracting the prompt-dependent baseline leaves the population MaxRL gradient in Section 3.1 unchanged. The finite-sample updates nevertheless differ, and this on-policy identity does not apply unchanged to responses generated by an earlier policy.

Training speed. Over 12000 updates, the production logs record approximately 3.4 seconds per update for unified training at 0% mixing and 28.8 seconds for staged RL, an $8.4\times$ difference in update throughput. These timings exclude launch and restart downtime and correspond to the implemented systems, which use 24 and 16 GPUs, respectively. The asynchronous trainer can reuse available responses across updates, so this comparison measures optimization throughput rather than the rate of generating fresh rollouts.

Task performance and retention. Figure 6 shows similar mathematical accuracy for zero mixing and staged RL: Math pass@1 is 24.31% and 24.16%, respectively. Staged RL has slightly higher mathematical coverage, with pass@32 of 68.56% compared with 67.64%, and a smaller increase in held-out corpus loss, 0.047 versus 0.104 nats/token. This comparison reflects the complete implementations, including the optimizer settings in Section D.1, and does not isolate the effects of staleness or baseline subtraction individually.

D.7 Coding RL after Mathematical Training

Initialization. We continue training on coding tasks using the staged mathematical model and four unified models trained with constant midtraining proportions of 0.5, 1, 2, and 5%. Each model starts from its final mathematical-training checkpoint. Curve labels identify the models' mathematical training histories; all subsequently use the same coding RL configuration without midtraining data. The comparison therefore tests how the preceding training affects adaptation to coding.

Data and training. We use 20000 Python problems from KodCode-V1 (Xu et al., 2025),[5] comprising 10000 labeled medium difficulty and 10000 labeled hard difficulty. The reward is one if the generated program passes all supplied unit tests and zero otherwise. All models use the shared training settings in Table 17.

Setting	Value
Algorithm	On-policy MaxRL with reward-baseline subtraction
Optimizer	Fresh AdamW
Adam (β_1, β_2); ε	$(0.9, 0.999)$; 10^{-8}
Weight decay; gradient clipping norm	0; 1
Learning rate / schedule	10^{-6} / constant, no warmup
Prompts per update	32
Responses per prompt	16
Global batch size	512 responses
Microbatch per device	16 responses
Prompt + response limit (tokens)	1024
Temperature / top-k / top-p	1 / 50 / 0.95
KL penalty coefficient	0

Table 17: **Coding adaptation hyperparameters.** All initializations use the same coding RL settings.

Evaluation. Figure 8 reports pass@1 and pass@16 at coding updates 0, 1000, 2000, 3000, and 4000. We sample 16 responses for each of the 164 HumanEval+ and 378 MBPP+ problems, using the EvalPlus tests and decoding settings in Section D.1. Step zero reuses the recorded evaluation of each initialization checkpoint; subsequent points evaluate the corresponding coding checkpoints. We also track MATH-500 and GSM8K pass@1 and pass@16 using 16 responses per problem, MMLU accuracy using zero-shot answer-choice likelihoods, and SciQ and ARC-Challenge accuracy using greedy decoding. Held-out corpus loss uses the same 24-component average as in Section D.1.

Performance beyond coding. Figure 24 shows how mathematical performance and general knowledge change during coding RL. Mathematical accuracy remains close to its starting level, whereas pass@16 decreases on both benchmarks for every initialization. MMLU and SciQ accuracy also remain close to their starting values, preserving the unified models' advantage on these benchmarks. Held-out corpus loss increases during coding RL but remains lower for the unified models than for staged training.

[5] https://huggingface.co/datasets/KodCode/KodCode-V1

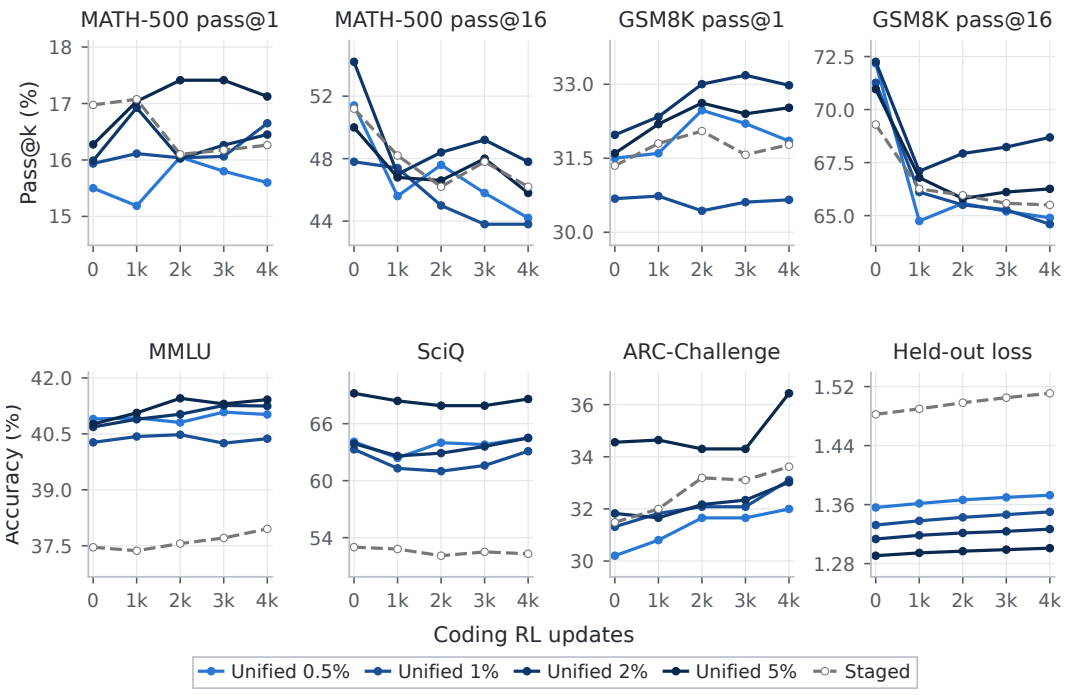


Figure 24: **Mathematical performance and retention during coding RL.** The initializations and coding updates are the same as in Figure 8. Percentages denote constant midtraining proportions during the preceding mathematical training. Pass@1, pass@16, and benchmark accuracy are in percent; held-out loss is in nats/token. Mathematical, MMLU, and SciQ accuracy remain close to their starting values, while mathematical coverage declines and held-out loss increases.

D.8 WebInstruct RL after Mathematical Training

Setup. To examine whether the continual-learning benefits extend beyond coding, we continue training on WebInstruct using the staged mathematical model and the unified models trained with constant midtraining proportions of 2 and 5%. These are the same mathematical checkpoints used in Section D.7, before coding RL. All three models use the same on-policy MaxRL procedure without midtraining data, with the hyperparameters in Table 17. We randomly sample 20000 nonmathematical training questions from WebInstruct-verified and its unfiltered release (Ma et al., 2026), covering fields such as physics, chemistry, business, and history. We reserve disjoint sets of 512 validation and 1000 test questions. The selected questions have numerical or free-form answers; decimal-number answer types are excluded. The reward is binary: the extracted final answer must be accepted by the General-Reasoner verifier and satisfy answer-format checks, with an additional 1% relative-error check for numerical answers.

Evaluation. We report training-time validation accuracy every 100 updates and test pass@1 and pass@64 at updates 0, 1000, 2000, and 3000. Validation accuracy is the fraction of successful responses across 16 samples for each of the 512 validation questions. Test evaluation uses 64 responses per question, with temperature 1, top-k 50, top-p 0.95, and up to 1024 response tokens within a 2048-token context. Three test prompts exceed the 1024-token prompt budget, leaving the same 997 questions for every checkpoint. We use the pass@k estimator in Section C.4 and report results separately for 502 questions with numerical answers (integers, percentages, and fractions) and 495 with free-form answers (strings, lists, and expressions).

Results. Figure 25 shows that both unified models begin with lower validation and test accuracy than staged training but achieve higher accuracy after 3000 updates. The 2% model also achieves higher test pass@64, whereas the 5% model remains slightly below staged training on this metric. The answer-type breakdown shows higher final free-form accuracy and coverage for both unified models than for staged training; on numerical answers, only the 2% model achieves higher coverage. These results provide further evidence that unified training can yield a better starting point for subsequent learning, with benefits that vary across answer types and metrics.

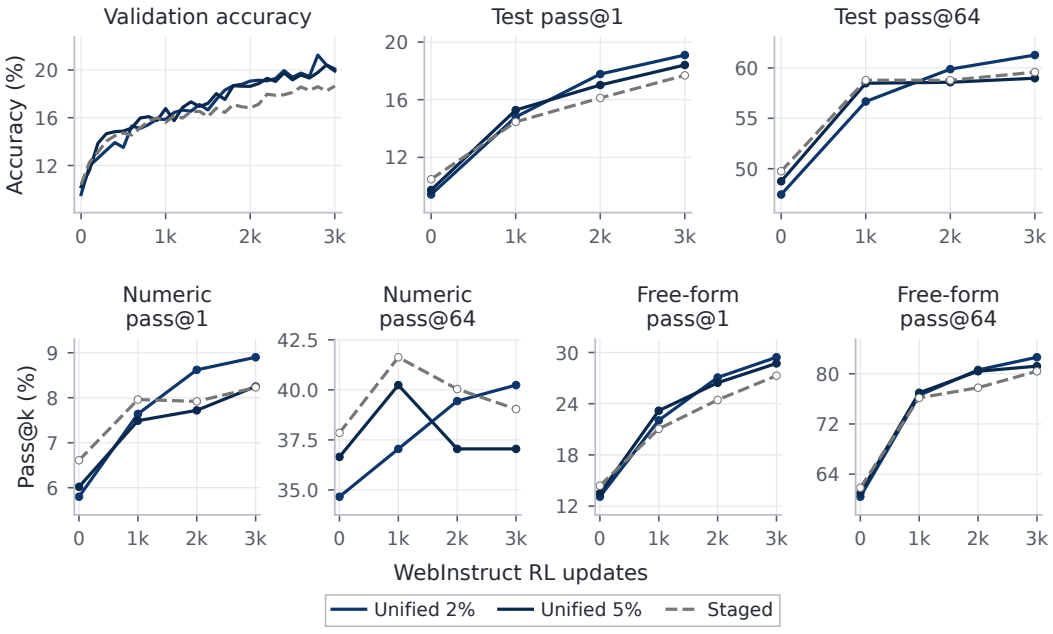


Figure 25: **WebInstruct RL after mathematical training.** Percentages identify the midtraining proportions during mathematical training; all models then use the same WebInstruct RL procedure without midtraining data. Top: validation accuracy and overall test performance. Bottom: test performance on numerical and free-form answers separately. All performance metrics are in percent.

E Comparison with KL Regularization

We examine whether the benefits of mixing midtraining data during RL can be reproduced by regularizing the policy toward the base model. Using the real-world data setting of Section 5.2, we compare unified training with mathematical RL augmented by a sampled KL penalty. This comparison tests whether constraining policy changes yields the same improvements as continued learning from midtraining data.

Experimental setup. The comparisons use the same base model, Math12K prompts, and benchmark protocol. Unified training is represented by the constant-mixture results in Figure 6, with midtraining proportions of 0.2, 0.5, 1, 2, and 5%. The KL comparison uses no midtraining data and varies the penalty coefficient over $\beta \in \{0, 0.001, 0.01, 0.1\}$.

KL penalty. The reference is the frozen base model from which training starts. At each sampled RL response token, the penalty is $e^u - u - 1$, where u is the reference log-probability minus the current-policy log-probability, clipped to $[-20, 20]$. Log-probabilities are computed over the full vocabulary. The penalty applies to both correct and incorrect responses and is added to the weighted token-loss numerator with coefficient β, using the same loss normalization as the RL term.

Evaluation. Benchmark metrics and the component-averaged held-out corpus loss follow Section D.1. Training-time validation instead measures token-averaged loss on a fixed set of 512 held-out midtraining windows, evaluated every 200 updates and at the final checkpoint. The benchmark comparison includes all three nonzero KL coefficients and all five unified mixtures. Benchmarks are evaluated every 1000 updates; MMLU and the component-averaged corpus loss are evaluated every 2000 updates. The validation curves additionally include the unregularized control ($\beta = 0$).

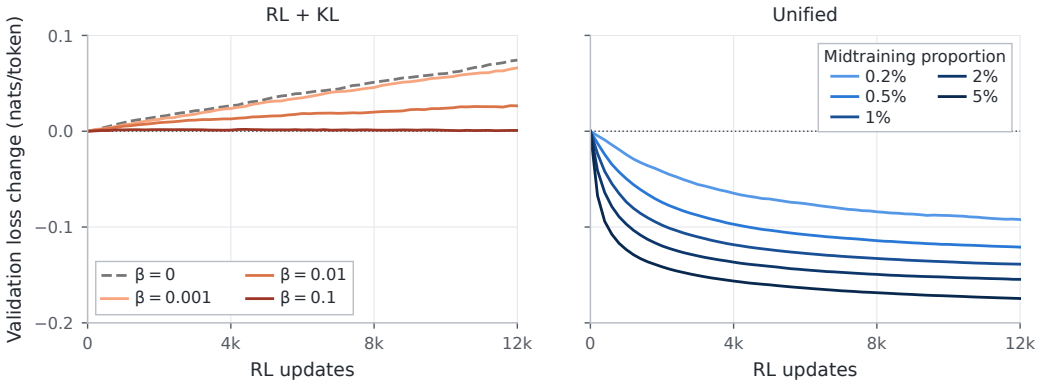


Figure 26: **Held-out loss during KL-regularized RL and unified training.** Each curve shows the change from the common base-model value in the trainer's midtraining validation loss. This training-time metric is distinct from the component-averaged corpus loss described in Section D.1. Curves show recorded evaluations without smoothing.

Results. Increasing the KL coefficient reduces the deterioration in held-out loss, but the strongest penalty ($\beta = 0.1$) also substantially reduces mathematical accuracy and lowers coverage relative to the weaker penalties (Figs. 26 and 27). Unified training reduces held-out loss below the base-model value while achieving mathematical accuracy and coverage similar to the weaker KL controls. At the final checkpoints, all five unified mixtures also achieve higher SciQ and MMLU accuracy than any of the KL-regularized runs.

The comparison still involves tradeoffs: the weaker KL penalties yield higher coding accuracy, while coding coverage and ARC-Challenge accuracy show no consistent advantage for either approach. Nevertheless, none of the tested KL penalties reproduces the combination of mathematical performance, lower corpus loss, and higher SciQ and MMLU accuracy obtained with unified training. These results indicate that the benefits of mixing midtraining data extend beyond those obtained by regularizing toward the base policy in this setting.

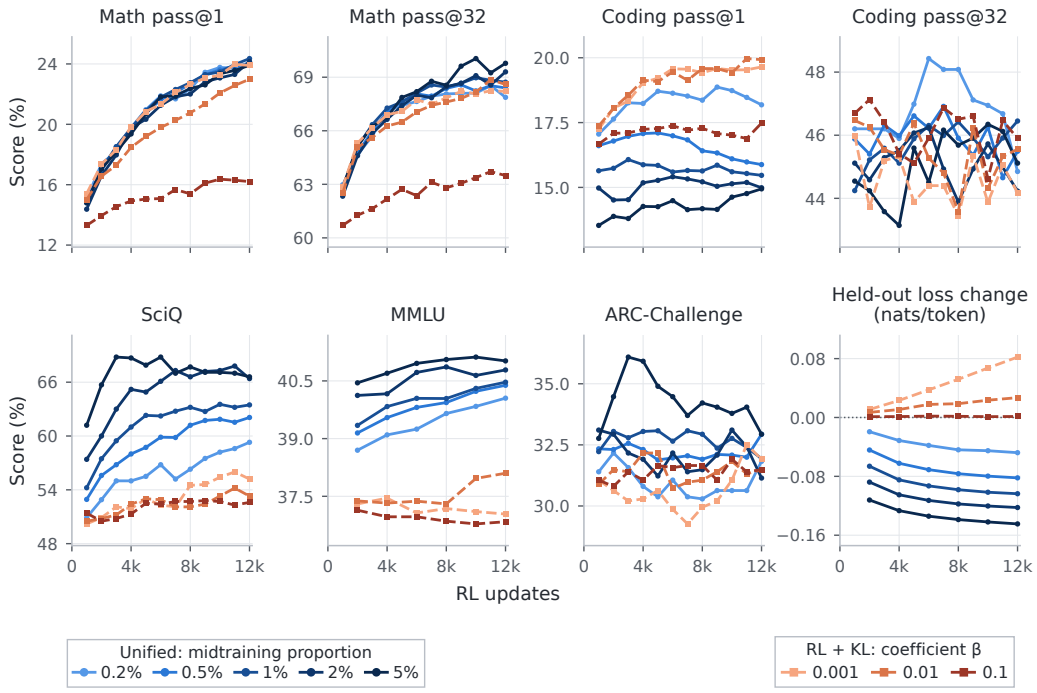


Figure 27: **Benchmark performance during KL-regularized RL and unified training.** Solid blue curves show unified training at five midtraining proportions; dashed orange curves show RL with three KL coefficients. Markers indicate recorded evaluations, joined without smoothing. Math and Coding average the two benchmarks in each domain, as defined in Section D.1. Math pass@32 uses 256 responses per problem, and Coding pass@32 uses a separate 32-response evaluation. Held-out loss change uses the component-averaged corpus loss, with negative values indicating improvement over the base model.

F EXAMPLES OF GRPO COLLAPSE

We report two examples of unstable GRPO training on Reasoning Gym that motivated our use of on-policy MAXRL for the staged baselines. In both runs, initial improvements are followed by a collapse of both accuracy and coverage to approximately zero, which persists through the end of training.

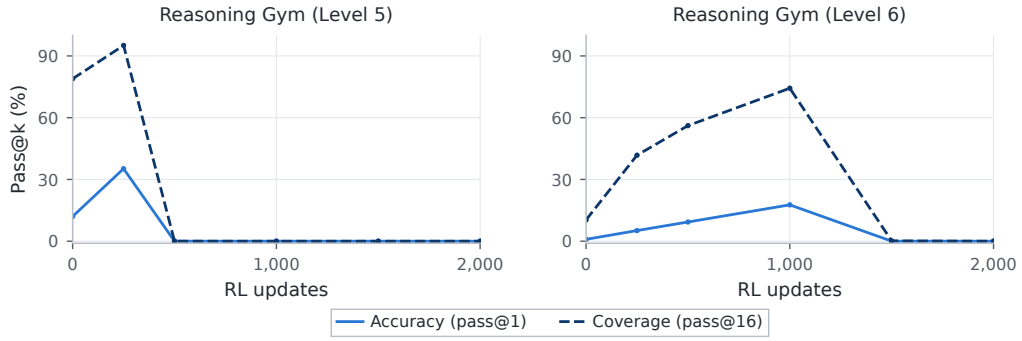


Figure 28: **Examples of GRPO collapse in staged training.** The panels show Reasoning Gym runs at Levels 5 and 6. Solid curves show accuracy (pass@1), and dashed curves show coverage (pass@16). Each panel shows the complete recorded evaluation history through the configured training budget, without smoothing.

Reasoning Gym. These auxiliary runs use the same 124M-parameter model and 20B-token pretraining corpus as Section 5.1, followed by four epochs of midtraining on an earlier 25600-trace reasoning corpus at a learning rate of 2×10^{-4}. We then train separately on Levels 5 and 6 for 2000 GRPO updates at a learning rate of 10^{-5}, with 32 prompts and 16 responses per prompt. Evaluation uses 1024 held-out problems per level at temperature 1.0, with 64 samples per question at RL checkpoints and 16 at initialization. The training reward also falls to zero in both runs, consistent with the collapse in held-out performance.

G Policy Gradient Objectives and Continuous Rewards

This appendix derives the extensions in Section 3.2 to REINFORCE and TailRL. Both fit the common likelihood loss $\mathcal{L}(\theta; q)$ by choosing an appropriate joint data distribution. We distinguish throughout between constructing a target from a generation policy and differentiating the loss with that target held fixed. We assume that the objectives are finite and that differentiation can be exchanged with the expectations and integrals below. Table 18 summarizes the corresponding targets and effective prompt distributions for on-policy updates.

Method	Reward	Target μ_s	Effective prompts ρ_s
MaxRL	Binary	$\pi^+_{\theta_t}$	$\rho(x)$
REINFORCE	Binary	$\pi^+_{\theta_t}$	$\propto \rho(x) p_{\theta_t}(x)$
TailRL	$[0,1]$	$\int_0^1 \pi_{\theta_t}(\cdot \mid x, r > \tau)\, d\tau$	$\rho(x)$
REINFORCE	$[0,1]$	$\frac{\pi_{\theta_t}(y \mid x) r(x,y)}{\mathbb{E}_{y' \sim \pi_{\theta_t}(\cdot \mid x)}[r(x,y')]}$	$\propto \rho(x) \mathbb{E}_{y \sim \pi_{\theta_t}(\cdot \mid x)}[r(x,y)]$

Table 18: RL objectives within the unified likelihood framework. Different objectives correspond to different target and effective prompt distributions.

G.1 REINFORCE

For binary rewards, REINFORCE and MaxRL fit the same distribution of correct completions, but weight prompts differently. The REINFORCE construction also applies to nonnegative continuous rewards. Let ν be a fixed generation policy, and write $p_\nu(x) := \mathbb{E}_{y \sim \nu(\cdot \mid x)}[r(x,y)]$ for its expected reward. When this expectation is finite and positive, the reward-weighted target is

$$\nu^+(y \mid x) = \frac{\nu(y \mid x) r(x,y)}{p_\nu(x)}.$$

For continuous rewards, this target assigns probability in proportion to both the generation probability and the reward. For binary rewards, it is simply ν conditioned on correctness.

The likelihood update. Weighting each prompt's loss by its expected reward gives

$$\mathcal{L}_{\mathrm{RF}}(\theta; \rho \circ \nu) := -\mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \nu^+(\cdot \mid x)}[p_\nu(x) \log \pi_\theta(y \mid x)]. \tag{3}$$

Since ν^+ and p_ν are fixed, only the model log-likelihood is differentiated:

$$\begin{aligned} -\nabla_\theta \mathcal{L}_{\mathrm{RF}}(\theta; \rho \circ \nu) &= \mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \nu^+(\cdot \mid x)}[p_\nu(x) \nabla_\theta \log \pi_\theta(y \mid x)] \\ &= \mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \nu(\cdot \mid x)}[r(x,y) \nabla_\theta \log \pi_\theta(y \mid x)]. \end{aligned}$$

The second equality uses $p_\nu(x)\nu^+(y \mid x) = \nu(y \mid x) r(x,y)$. Taking $\nu = \pi_{\theta_t}$ and evaluating the gradient at θ_t recovers the REINFORCE policy gradient in Eq. (1). If $p_\nu(x) = 0$, the reward-weighted update is zero, so such a prompt contributes no gradient and requires no normalized target.

Absorbing the prompt weights. The factor $p_\nu(x)$ changes how frequently each prompt contributes to training. It can therefore be represented by a change in the prompt distribution. Define

$$Z := \mathbb{E}_{x \sim \rho}[p_\nu(x)], \qquad \rho_{\mathrm{RF}}(x) := \frac{\rho(x) p_\nu(x)}{Z},$$

where $0 < Z < \infty$. Then Eq. (3) is exactly $Z\mathcal{L}(\theta; \rho_{\mathrm{RF}} \circ \nu^+)$. Thus, REINFORCE uses the original training objective with a different data distribution. The scalar Z preserves its gradient magnitude and must be retained in the source weight when mixing RL with static data.

Baselines and reward normalization. A fixed prompt-dependent baseline $b_t(x)$ contributes zero to the expected on-policy update, since

$$\mathbb{E}_{y \sim \pi_{\theta_t}(\cdot \mid x)}[\nabla_\theta \log \pi_\theta(y \mid x)]_{\theta = \theta_t} = 0.$$

Consequently, rewards with a finite lower bound $b(x)$ independent of θ can be shifted to $r(x,y) - b(x) \geq 0$ without changing the population REINFORCE gradient. The construction above then applies to the shifted reward. This baseline invariance need not hold for an uncorrected update using data from an earlier policy.

For nonnegative rewards, dividing $r(x,y) - b_t(x)$ by a fixed $s_t(x) > 0$ gives the same expected on-policy update as fitting $\pi^+_{\theta_t}$ with prompt weight $p_{\theta_t}(x)/s_t(x)$. Population standard-deviation normalization for binary rewards gives weight $\sqrt{p_{\theta_t}(x)/(1-p_{\theta_t}(x))}$ for $0 < p_{\theta_t}(x) < 1$, as in the population analysis of GRPO (Tajwar et al., 2026). These are identities for population updates. Empirical advantages, policy ratios, clipping, and KL penalties in PPO and GRPO need not preserve the fixed-target likelihood formulation (Schulman et al., 2017; Shao et al., 2024).

G.2 Continuous Rewards with TailRL

TailRL (Ramasubramanian et al., 2026) extends MaxRL to continuous rewards by applying its likelihood objective at each reward threshold. For rewards $r(x,y) \in [0,1]$, a threshold τ defines the binary event $r(x,y) > \tau$. Write its probability under the model as

$$q_\theta(x,\tau) := \Pr_{y\sim\pi_\theta(\cdot|x)}(r(x,y) > \tau).$$

TailRL averages the log-probability of this event over a uniformly chosen threshold:

$$J_{\mathrm{Tail}}(\theta) := \mathbb{E}_{x\sim\rho}\left[\int_0^1 \log q_\theta(x,\tau)\, d\tau\right].$$

To connect this objective to unified training, we first construct a target at each threshold and then combine these targets into a single distribution.

A target at each reward threshold. Fix a generation policy ν, and let $q_\nu(x,\tau)$ denote its corresponding tail probability. Assume these probabilities are positive for $\tau < 1$ and that the TailRL objective is finite. Conditioning on the event $r(x,y) > \tau$ gives

$$\nu^+_\tau(y \mid x) := \frac{\nu(y \mid x)\, \mathbf{1}\{r(x,y) > \tau\}}{q_\nu(x,\tau)}.$$

This is the success-conditioned target construction in Section 3.2, with success now defined by the selected threshold.

Combining the targets. Because TailRL averages uniformly over thresholds, the corresponding target is the uniform mixture

$$\nu_{\mathrm{Tail}}(y \mid x) := \int_0^1 \nu^+_\tau(y \mid x)\, d\tau.$$

Each ν^+_τ is normalized, so this mixture is also a probability distribution over completions. The threshold is used only to construct the target; the model still conditions on the original prompt x. By linearity of expectation, fitting the mixture gives the average of the likelihood gradients for the individual thresholds:

$$\begin{aligned} -\nabla_\theta \mathcal{L}(\theta; \rho \circ \nu_{\mathrm{Tail}}) &= \mathbb{E}_{x\sim\rho}\, \mathbb{E}_{y\sim\nu_{\mathrm{Tail}}(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)] \\ &= \mathbb{E}_{x\sim\rho}\left[\int_0^1 \mathbb{E}_{y\sim\nu^+_\tau(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)]\, d\tau\right]. \end{aligned}$$

Taking $\nu = \pi_{\theta_t}$ and evaluating at θ_t recovers $\nabla_\theta J_{\mathrm{Tail}}(\theta_t)$. Thus, a TailRL update is obtained from the same loss $\mathcal{L}$ with joint data distribution $q = \rho \circ \nu_{\mathrm{Tail}}$. When the generation policy is an earlier model, the update instead fits that policy's mixture of conditional targets, just as in the binary-reward construction in Section 3.2.

Binary rewards and finite samples. For binary rewards, every threshold $\tau \in [0,1)$ selects exactly the correct completions. Hence $\nu^+_\tau = \nu^+$ at every such threshold, and their mixture is also ν^+. Both the objective and the target construction reduce to MaxRL.

The derivation above concerns the population target and gradient. The MaxRL weighting described in Section 4 is specific to binary rewards; estimators for continuous rewards must account for the contributions of different thresholds. Ramasubramanian et al. (2026) derive these estimators and the finite-order TailRL objectives whose gradients they estimate.