TailSFT: Filtered Fine-Tuning Improves Post-Training Performance

Sadhika Malladi[†] Samy Jelassi[‡] Dylan J. Foster[‡] Jordan T. Ash[‡] Akshay Krishnamurthy[‡]

Abstract

Reinforcement learning post-training drives reasoning and agentic capabilities in modern AI systems, yet a growing body of work shows that it is most effective when used to fine-tune an already capable base model. We question whether existing pipelines yield models that are most suitable for reinforcement learning. Building on prior work highlighting the role of *coverage* and pass@K as predictors of post-RL performance, we design a simple modification to supervised fine-tuning, *TailSFT*, which filters out already fit sequences during training, thereby focusing learning on under-modeled regions, or the tail, of the data distribution. We justify and validate the design choices in TailSFT, particularly the specific filtering criteria, through a combination of controlled experiments and theoretical analysis. On OLMo-3 7B, TailSFT often improves pass@16 performance on math and coding evaluations, with gains up to 17% absolute, while incurring minimal computational overhead. These higher-coverage checkpoints consistently translate to up to 4% absolute pass@1 gains in subsequent GRPO runs, demonstrating that TailSFT checkpoints serve as better initializations for RL. We further introduce a lightweight diagnostic for identifying settings where TailSFT is most likely to help. More broadly, our results motivate a principled, stage-aware approach to model development, in which intermediate checkpoints are judged by how effectively they support subsequent training.

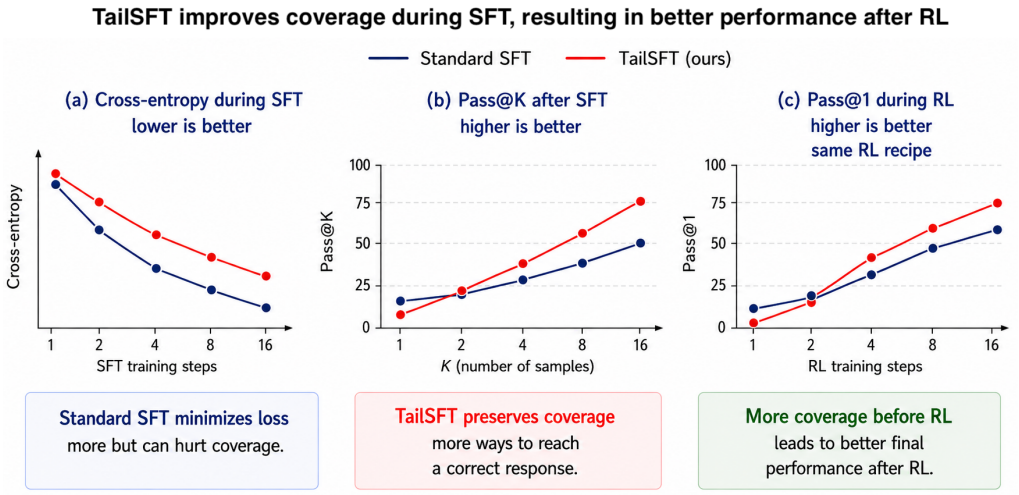

Figure 1: **TailSFT yields better coverage, possibly at the cost of worse cross-entropy, which translates to stronger post-RL performance.** Standard SFT achieves lower cross-entropy, while TailSFT preserves higher pass@K at large K. Starting from this higher-coverage checkpoint, the same RL procedure yields higher pass@1. Curves are illustrative.

[†]University of California San Diego, sadhika.malladi98@gmail.com. Work partially completed while at Microsoft Research NYC.
[‡]Microsoft Research NYC, {samyjelassi,dylanfoster,ash.jordan,akshaykr}@microsoft.com.

1 Introduction

Language models are commonly trained in multiple phases, each defined by a different dataset and objective. A typical pipeline includes (1) next-token prediction on a general-purpose corpus, (2) supervised fine-tuning (SFT) for instruction following and domain adaptation, and (3) reinforcement learning (RL) for complex reasoning tasks. A tacit assumption in such pipelines is that improving the objective at one stage also produces a better initialization for the next. Recent work suggests that this assumption need not hold: models that perform better according to the local objective of one stage can nevertheless be worse starting points for subsequent training (Zhang et al., 2026a; Wang et al., 2025; Springer et al., 2025; Chen et al., 2025). These findings motivate a stage-aware view of model development, in which an intermediate model is judged not only by its standalone performance, but also by how effectively it supports the training that follows.

This issue is particularly important at the transition from SFT to RL. RL has produced substantial gains in mathematics, code, and other reasoning domains (Shao et al., 2024; Guo et al., 2025; MAI, 2025; Zhao et al., 2026a; Kimi, 2026), but requires substantial computation to generate and evaluate on-policy responses. In RL with verifiable rewards (RLVR), the usefulness of these rollouts depends strongly on the initialization. If rewarding responses are difficult to sample from the SFT model, many rollout groups contain little direct positive learning signal; conversely, if rewarding responses are reachable under repeated sampling, RL has more useful behavior to reinforce. Thus, designing SFT for subsequent RL requires a criterion that captures not merely single-sample accuracy, but whether useful responses remain accessible within the rollout budget available to RL.

The *coverage principle* formalizes this desideratum by relating a model's coverage profile to what repeated sampling can recover (Chen et al., 2026a) (Definition 2.1). Empirically, pass@k provides a convenient measure of this property, in that it quantifies how often k samples contain at least one reward-bearing response. At values of k comparable to the sampling budget used during RL, large-k pass@k therefore measures how frequently the initialization exposes RL to a useful response.

Standard SFT, however, is not explicitly designed to preserve or improve this form of coverage. Cross-entropy continues to reward increases in the likelihood of every demonstration, including responses that the model can readily produce already. Further fitting these responses can shift probability mass away from potentially useful responses represented by the initial model. Consequently, cross-entropy and coverage need not be aligned. A model with worse cross-entropy can have better coverage, and vice versa (Chen et al., 2026a, 2025). On the other hand, directly optimizing the coverage profile is difficult, in part because the criterion is non-differentiable and depends on information about the target response distribution that is generally unavailable (Definition 2.1). This raises the question we study in this work:

Can we modify supervised fine-tuning so that it better preserves the useful response coverage needed for subsequent reinforcement learning?

Contributions. To better prepare models for RL post-training, we propose TailSFT, an SFT algorithm that prioritizes coverage (or high pass@k at large k) over low cross-entropy loss (Algorithm 1). TailSFT is a sequence-level filtering method designed to direct training effort towards learning the under-modeled tail of the response distribution. The algorithm is lightweight and serves as a straightforward drop-in objective for SFT.

1. We isolate the effect of filtering in a controlled graph navigation task (Section 3.2) and show that filtering already-fit examples improves coverage, achieving higher pass@k at large k despite worse cross-entropy and pass@1 (Figures 2 and 3).
2. We design TailSFT to filter relative to the initial policy and show that this criterion provably ensures better coverage than standard cross-entropy or more naïve filtering (Theorem 3.1).
3. We apply TailSFT to OLMo-3 7B on standard math and coding tasks and obtain substantial gains in coverage and post-RL performance. TailSFT improves pass@16, in absolute terms, by up to 16.8% on coding and 3.1% on math (Table 1), and improves final pass@1 after GRPO by up to 3.9% (Table 2).
4. We introduce a lightweight coverage-ratio diagnostic (Definition 4.1), computed from the base model and a standard SFT run, that provides a sufficient condition for TailSFT to improve coverage (Figure 4).

Broader outlook. Our results illustrate a broader source of suboptimality in multi-stage model training, where the objective that is appropriate for producing a strong model at one stage may not be the objective that produces the best initialization for the next. For the SFT-to-RL transition studied here, this distinction means that high large-k pass@k can be more valuable than lower cross-entropy, even when the two criteria are in tension. Recent work has documented related forms of misalignment across pre-training, fine-tuning, and RL (Zhang et al., 2026a; Wang et al., 2025; Springer et al., 2025; Chen et al., 2026a, 2025; Watts et al., 2026). TailSFT complements these observations with a practical intervention: rather than only diagnosing whether an intermediate model will support later training, it modifies SFT itself to produce a more useful initialization for the RL stage. Based on our results, we believe there is significant potential in adopting a holistic approach to language modeling.

2 Preliminaries on Coverage

We formalize coverage as a property of the SFT initialization that captures how readily reward-bearing responses can be sampled. We then relate coverage to pass@K, which provides an empirical measure of the learning signal available to RL at the start of post-training.

Notation and setting. Consider a prompt distribution μ over a prompt space $\mathcal{X}$, a response space $\mathcal{Y}$, and a language model $\pi(\cdot \mid x) \in \Delta(\mathcal{Y})$. We focus on verifiable tasks with binary reward $R : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, where $R(x, y) = 1$ indicates that response y is correct for prompt x. In SFT, we observe (x_i, y_i) with $x_i \sim \mu$ and $y_i \sim \pi_{\mathrm{D}}(\cdot \mid x_i)$, where π_{D} assigns probability to reward-bearing responses. Standard SFT minimizes the sequence-level cross-entropy $\ell_\pi(x, y) = -\log \pi(y \mid x)$, and the resulting model π_{SFT} initializes RL.

The coverage profile. Consider a single prompt x, and let $p_\pi(x) := \Pr_{y\sim\pi(\cdot|x)}[R(x, y) = 1]$. A batch of K independent rollouts contains at least one rewarding response with probability $1 - (1 - p_\pi(x))^K$. If all K rollouts receive zero reward, no gradient is received for their corresponding prompt. Thus, useful responses must have sufficient probability under the initialization to have a chance of being produced during RL. Across a dataset of many prompts, the model may benefit via positive transfer from easier to harder prompts, but the initial probability of rewarding responses, $p_\pi(x)$, determines the learning signal available before such transfer.

The coverage profile formalizes this property relative to the ground-truth, SFT-data policy (Chen et al., 2026a). We assume that π_{D} assigns non-negligible probability to reward-bearing responses, so coverage of π_{D} is relevant to downstream RL.

Definition 2.1 (Coverage Profile)**.** *Given a data-generating policy π_{D} and model π, the coverage profile at scale N is*

$$\mathrm{Cov}_N(\pi_{\mathrm{D}} \| \pi) := \Pr_{x\sim\mu,\, y\sim\pi_{\mathrm{D}}(\cdot|x)} \left[\frac{\pi_{\mathrm{D}}(y \mid x)}{\pi(y \mid x)} \geq N \right].$$

The coverage profile measures the probability mass under π_{D} that π underweights by a factor of at least N; smaller values indicate better coverage. The scale N determines the sampling budget needed to reach this mass. Chen et al. (2026a) show that, if $\mathrm{Cov}_N(\pi_{\mathrm{D}} \| \pi) \leq \frac{1}{2}$, then $K \geq 2N \log(1/\varepsilon)$ samples suffice for Best-of-K to achieve expected reward within $\mathrm{Cov}_N(\pi_{\mathrm{D}} \| \pi) + \varepsilon$ of π_{D}, with a corresponding worst-case necessity result with a comparable relationship between N and K. Thus, coverage at scale N characterizes which useful responses are accessible with a sampling budget on the order of N.

Since $\pi_{\mathrm{D}}(y \mid x)$ is generally unknown, the coverage profile cannot be measured directly. However, for binary rewards, pass@$K(\pi) := \mathbb{E}_{x\sim\mu}[1 - (1 - p_\pi(x))^K]$ is exactly the expected reward of Best-of-K sampling (Brown et al., 2024). We therefore use pass@K at large K as an empirical measure of coverage. At a K comparable to the RL rollout budget, it also measures how often the policy exposes RL to reward-bearing responses.

3 TailSFT: Supervised Fine-Tuning for Coverage

In this section, we develop TailSFT, a sequence-level filtering method designed to direct SFT updates toward improving coverage. The method combines two ideas: stopping updates on responses that are already sufficiently

Algorithm 1 TailSFT

Require: Initial policy π_0, SFT data $\mathcal{D}$, filtering schedule $\{\gamma_t\}$

Record $\ell_i^0 \leftarrow \ell_i(\pi_0)$ for every $(x_i, y_i) \in \mathcal{D}$

for each training step t **do**

  Sample a batch $\mathcal{B}_t$

  Compute $\ell_i^t \leftarrow \ell_i(\pi_t)$ for each $(x_i, y_i) \in \mathcal{B}_t$

  Let $\mathcal{F}_t \subseteq \mathcal{B}_t$ be the γ_t fraction of sequences with the smallest $\ell_i^t - \ell_i^0$

  Set π_{t+1} by taking a gradient step from π_t on

$$\mathcal{L}_t(\pi) = \frac{\sum_{i \in \mathcal{B}_t \setminus \mathcal{F}_t} -\log \pi(y_i \mid x_i)}{\sum_{i \in \mathcal{B}_t \setminus \mathcal{F}_t} |y_i|}$$

well modeled via filtering, and determining which responses to filter relative to the initial policy. We first introduce the algorithm, then isolate the effect of filtering in a controlled graph-navigation task, and finally analyze why the initial policy provides a useful reference for filtering.

3.1 The TailSFT Algorithm

Coverage at scale N depends on whether useful responses receive enough probability to be reached via repeated sampling. Once a response is already well covered at that scale, increasing its likelihood further does not reduce Cov_N. Improving coverage instead requires increasing the probability of responses that remain under-modeled. Standard SFT makes no such distinction and continues to increase the likelihood of every training response.

TailSFT reduces training on responses that have already improved substantially and concentrates subsequent updates on the remaining tail. Since $\pi_{\mathrm{D}}(y \mid x)$ is unknown, we cannot determine directly which responses are already covered. Instead, we compare each response's current loss to its loss under the initial policy and filter the responses whose losses have decreased the most.

As described in Algorithm 1, TailSFT redirects training away from responses whose losses have decreased the most and toward responses that remain under-modeled. Filtering is applied at the sequence level. The length-normalized loss is used only to determine which sequences are filtered; optimization uses the standard token-averaged cross-entropy over target tokens in the retained sequences. The filtering fraction γ_t may be fixed or vary over training.

Filtering itself does not require using the initial policy as a reference. We therefore consider two alternatives alongside TailSFT: *absolute filtering*, which stops training on an example once its loss falls below a fixed threshold, and *quantile filtering*, which filters the lowest-loss fraction of each batch. These alternatives allow us to separate the benefit of filtering itself from the benefit of measuring progress relative to the initial policy π_0.

3.2 Warm-Up: Graph Navigation

We study whether halting updates on an example once it is fit by the model can improve coverage, even at the cost of the standard SFT objective and pass@1. To do so, this section considers a controlled setting in which the distribution of rewarding responses is known exactly. Following Chen et al. (2026a), each prompt x specifies a layered directed graph with source s, target t, and exactly eight valid $s \to t$ paths. Pretraining pairs each graph with a uniformly sampled valid path, while the SFT data deterministically selects and rewards a single path (see Figure 2). Thus, pretraining teaches generic graph navigation, while SFT teaches the task-specific path-selection rule. Since the target policy is deterministic, pass@K is exactly the probability that the rewarded path is contained in K samples, yielding a finite-sample estimate of coverage. Full details of the graph construction and data-generating process appear in Appendix C.

Using a GPT-2-style architecture and a shared pretraining setup, we vary only the SFT objective. We compare standard SFT with absolute filtering, quantile filtering, and TailSFT, which uses offset quantile filtering (as in Algorithm 1). For each filtering variant, we sweep its single filtering hyperparameter and report the setting with the highest final pass@8; complete training details and sweeps appear in Appendix C.

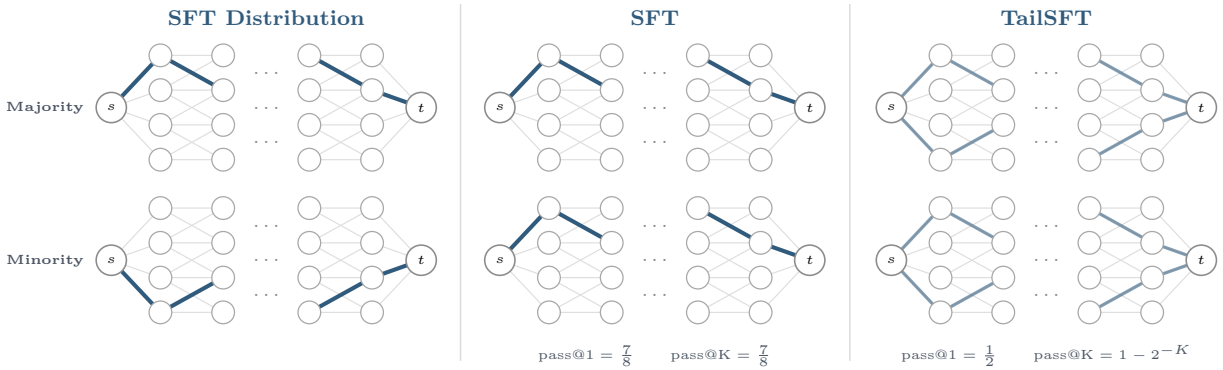

Figure 2: **TailSFT trades single-sample accuracy for better coverage under repeated sampling.** The SFT distribution contains majority and minority shards with different rewarding paths. Standard SFT over-anchors on the majority shard, achieving high pass@1 but no improvement as K grows. A filtered objective, illustrated here by TailSFT, retains probability on both paths, yielding lower pass@1 but substantially better pass@K scaling.

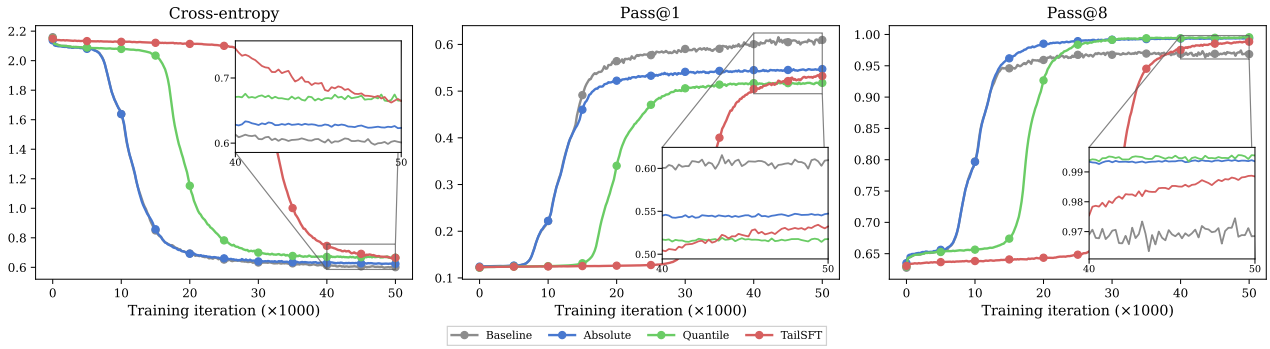

Figure 3: **Filtering improves pass@K at the expense of cross entropy and pass@1.** Results on the graph-navigation task. Standard SFT achieves the lowest cross-entropy and highest pass@1, while all three filtering methods achieve substantially higher pass@8. See Appendix C for details.

The results in Figure 3 reveal a clear divergence between the standard SFT objective and coverage. Standard SFT achieves the best cross-entropy and pass@1, yet all three filtering methods achieve substantially better pass@8, which we use as a surrogate for coverage. Thus, improving cross-entropy and single-sample accuracy can come at the expense of the probability that a rewarded response appears within a finite rollout budget. Filtering mitigates this failure mode by reducing further updates on responses that are already well modeled.

The graph task therefore provides clean evidence for the first design choice underlying TailSFT: filtering already-fit examples can improve coverage. It is, however, too simple a setting to determine whether the filtering criterion should depend on the initial policy. In fact, TailSFT is slightly weaker than other filtering variants in this setting, due to homogeneity across prompts. Under the ideal pretrained policy, every prompt has the same possible loss reduction (i.e., $\log 8$), so the reference loss provides no information for distinguishing which examples have more room to improve. Imperfections in pretraining therefore introduce noise into the offset score without providing useful signal. We discuss this effect further in Appendix C.

3.3 Theoretical Analysis

We next turn to a setting where the initial policy carries information about the response distribution, and ask whether filtering rules can preserve it. To study this question, we turn to theoretical analysis. Let S denote the set of rewarding responses, and suppose the target policy is obtained by discarding all responses outside S and renormalizing the initial policy:

$$\pi^\star(y) \propto \pi_0(y)\mathbf{1}\{y \in S\}.$$

Thus, among rewarding responses, the target preserves the relative preferences already present in π_0. The initial model may place too much total probability on unrewarding responses, but it still contains information about how probability should be distributed among the rewarding ones.

We compare two ways to decide when an example should stop contributing to the SFT objective. Let $\ell_\pi(x, y) = -\log \pi(y \mid x)$ denote the sequence-level SFT loss. With an absolute filtering constant α, we use

$$\ell_{\text{abs}}(\pi; x, y) = \left[\ell_\pi(x, y) + \log(\alpha)\right]_+.$$

An example stops contributing once $\pi(y \mid x) \geq \alpha$. Because the same threshold is applied to every response, this criterion ignores how likely the response was under the initial policy.

TailSFT instead measures progress relative to the initial policy π_0. The corresponding offset loss is

$$\ell_{\text{off}}(\pi; x, y) = \left[\ell_\pi(x, y) - \ell_{\pi_0}(x, y) + \log \beta\right]_+,$$

which stops updating a response once

$$\pi(y \mid x) \geq \beta \pi_0(y \mid x).$$

The stopping point therefore adapts to the probability that π_0 already assigns to each response rather than imposing a common final threshold.

Theorem 3.1 (Informal)**.** *Let π_{ERM}, $\pi_{\text{ABS},\alpha}$, and $\pi_{\text{OFF},\beta}$ denote the policies obtained using standard SFT, absolute filtering, and offset filtering, respectively. For any initial policy, rewarding set, and SFT sample, offset filtering can be tuned to achieve coverage no worse than standard SFT or the best absolute threshold:*

$$\inf_\beta \text{Cov}_N(\pi^\star \| \pi_{\text{OFF},\beta}) \leq \min\left\{\text{Cov}_N(\pi^\star \| \pi_{\text{ERM}}), \inf_\alpha \text{Cov}_N(\pi^\star \| \pi_{\text{ABS},\alpha})\right\}.$$

The inequality can be strict, and absolute filtering can be worse than standard SFT.

The theorem formalizes the role of the initial policy as more than merely an initialization. In this setting, π_0 contains useful information about how probability should be distributed among rewarding responses. Standard SFT can overwrite this structure by fitting the empirical frequencies of a finite SFT sample, while absolute filtering pushes observed responses toward a common probability threshold regardless of where they started. Offset filtering instead uses each response's initial probability to determine when further training is unnecessary. By preserving more of the useful structure already present in π_0, it can achieve strictly better coverage. The full statement and proof appear in Appendix B.

Connections to prior work. TailSFT is most closely related to direct coverage optimization and loss-based data filtering. Chen et al. (2025) introduce the direct coverage optimization loss

$$\ell_{\text{DCO}}(\pi; x, y) = -\log\left(1 - (1 - \pi(y \mid x))^K\right),$$

which directly optimizes pass@K when y is the final solution. For large K, its gradient becomes small once $\pi(y \mid x)$ is sufficiently large, making it a soft analogue for filtering without reference to the initial policy. Theorem 3.1 shows that, in our stylized setting, offset filtering can achieve coverage no worse than the best absolute threshold and can be strictly better. TailSFT is also related to RHO-LOSS and subsequent variants, though these methods were designed and tested primarily to improve pretraining efficiency (Mindermann et al., 2022; Thirukovalluru et al., 2024; Lin et al., 2024; Zhao et al., 2026b). These approaches prioritize examples or tokens according to their potential loss reduction. TailSFT, by contrast, is designed to improve coverage for a subsequent RL stage and applies filtering at the sequence level. See Appendix A for further discussion.

4 Language Model Experiments

The results in Section 3 suggest that TailSFT can preserve reward-bearing responses more effectively than standard SFT. We now evaluate whether this behavior carries over to pretrained language models. We first compare standard SFT and TailSFT on math and code tasks. The variation across these results then allows us to study when preserving information from the base model is most useful. Finally, we follow matched SFT checkpoints through GRPO to test whether higher coverage before RL translates into stronger post-RL performance. Throughout, we use pass@16 as an empirical measure of coverage.

		pass@1			pass@16		
SFT data	Benchmark	Standard	TailSFT	Δ	Standard	TailSFT	Δ
OMI	AIME 2022–2025 (n=120)	3.65 $\pm$0.37	4.03 $\pm$0.02	+0.37	15.24 $\pm$2.02	18.31 $\pm$0.57	+3.07
	MATH-500 Level 5 (n=134)	24.80 $\pm$0.40	25.16 $\pm$0.40	+0.36	66.42 $\pm$0.00	69.15 $\pm$0.86	+2.74
	OMEGA-500 (n=500)	6.58 $\pm$0.50	6.51 $\pm$0.26	−0.06	32.80 $\pm$2.25	32.60 $\pm$1.56	−0.20
BigCode	MBPP+ (n=378)	53.04 $\pm$0.93	51.36 $\pm$0.14	−1.68	74.69 $\pm$1.10	78.84 $\pm$1.06	+4.14
	HumanEval+ (n=164)	45.06 $\pm$0.57	46.33 $\pm$1.07	+1.27	76.63 $\pm$0.70	80.08 $\pm$0.93	+3.46
	CruxEval-I (n=800)	29.79 $\pm$0.48	30.43 $\pm$0.15	+0.64	61.71 $\pm$2.27	65.79 $\pm$0.26	+4.08
	CruxEval-O (n=800)	4.16 $\pm$0.44	13.18 $\pm$0.30	+9.02	24.21 $\pm$2.38	41.00 $\pm$1.35	+16.79
	LiveCodeBench (n=612)	10.74 $\pm$0.46	11.49 $\pm$0.38	+0.75	30.39 $\pm$0.86	33.12 $\pm$0.34	+2.72
Magicoder	MBPP+ (n=378)	55.35 $\pm$0.47	54.60 $\pm$0.55	−0.75	75.31 $\pm$0.93	78.66 $\pm$1.25	+3.35
	HumanEval+ (n=164)	48.49 $\pm$0.58	47.69 $\pm$0.19	−0.80	77.85 $\pm$0.70	78.66 $\pm$1.83	+0.81
	CruxEval-I (n=800)	28.50 $\pm$0.13	26.48 $\pm$0.45	−2.02	59.75 $\pm$0.62	68.08 $\pm$0.94	+8.33
	CruxEval-O (n=800)	12.86 $\pm$0.19	16.16 $\pm$0.53	+3.30	38.08 $\pm$0.94	47.92 $\pm$1.66	+9.83
	LiveCodeBench (n=612)	14.84 $\pm$0.11	14.32 $\pm$0.38	−0.51	31.81 $\pm$0.34	33.22 $\pm$0.34	+1.42
OCI	MBPP+ (n=378)	58.26 $\pm$0.88	55.81 $\pm$0.06	−2.44	81.22 $\pm$0.75	82.36 $\pm$0.81	+1.15
	HumanEval+ (n=164)	54.76 $\pm$1.16	51.94 $\pm$1.83	−2.82	85.98 $\pm$1.61	83.23 $\pm$2.16	−2.74
	CruxEval-I (n=800)	29.46 $\pm$0.46	30.17 $\pm$0.64	+0.71	68.08 $\pm$1.77	71.58 $\pm$1.91	+3.50
	CruxEval-O (n=800)	9.30 $\pm$0.49	10.84 $\pm$0.37	+1.54	40.12 $\pm$0.00	44.46 $\pm$0.47	+4.33
	LiveCodeBench (n=612)	12.24 $\pm$0.36	11.49 $\pm$0.84	−0.75	34.59 $\pm$0.90	33.50 $\pm$0.16	−1.09

Table 1: **TailSFT improves coverage over standard SFT.** Across 18 model–benchmark pairs, TailSFT often improves pass@16, while changes in pass@1 are mixed. Values are percentages reported as mean $\pm$ standard deviation over three seeds; Δ denotes TailSFT minus Standard SFT in percentage points, shown in green when positive, red when negative, and yellow when within one standard deviation. We later establish a sufficient condition (Definition 4.1) that accurately predicts the failure modes of TailSFT exhibited in this table (Figure 4). MATH-500 Level 5 refers to the subset of the benchmark with the highest difficulty.

4.1 SFT Results

Setup. We fine-tune the OLMo-3 7B base model (Olmo, 2026) separately on domain-specific math and code instruction data, giving 18 dataset–benchmark pairs. For math, we train on a 350k-example subset of OpenMathInstruct-2 (OMI, Toshniwal et al. (2024)), decontaminated against MATH-500. We evaluate with the OLMES protocol (Gu et al., 2025) on the 2022–2025 AIME problems, the hardest level of MATH-500, and OMEGA-500. For code, we train separately on Magicoder (Wei et al., 2024b), BigCode Self-OSS-Instruct (Wei et al., 2024a), and OpenCodeInstruct (OCI, Ahmad et al. (2025)). We evaluate on MBPP+, HumanEval+, CruxEval-I/O, and LiveCodeBench. We focus on benchmarks where performance is not saturated.

We report pass@16 using the task-specific OLMES sampling protocol and average results over three seed sets. Results are reported in Table 1 and Appendix D.1 contains experimental details.

TailSFT improves coverage in most settings. Across the 18 dataset–benchmark pairs in Table 1, TailSFT improves pass@16 in 15 cases, while changes in pass@1 are mixed. OMEGA-500 is essentially unchanged, with an absolute difference of −0.20%. The largest gain occurs on CruxEval-O after SFT on BigCode, where pass@16 increases by an absolute 16.79%. With Magicoder, pass@16 increases by 8.33% on CruxEval-I and 9.83% on CruxEval-O. On AIME, it increases by 3.07%. On MBPP+, absolute gains are 4.14%, 3.35%, and 1.15% for BigCode, Magicoder, and OCI. Overall, improvements are substantially more consistent at large K than at $K = 1$, in line with the coverage motivation for TailSFT.

4.2 Coverage Ratio Diagnostic

The variation across these settings motivates a closer look at when TailSFT should perform well. In the graph-navigation task, standard SFT increased probability on the majority shard while reducing coverage of a rewarded path that was already represented by the initial policy. Based on this, we ask whether SFT in the

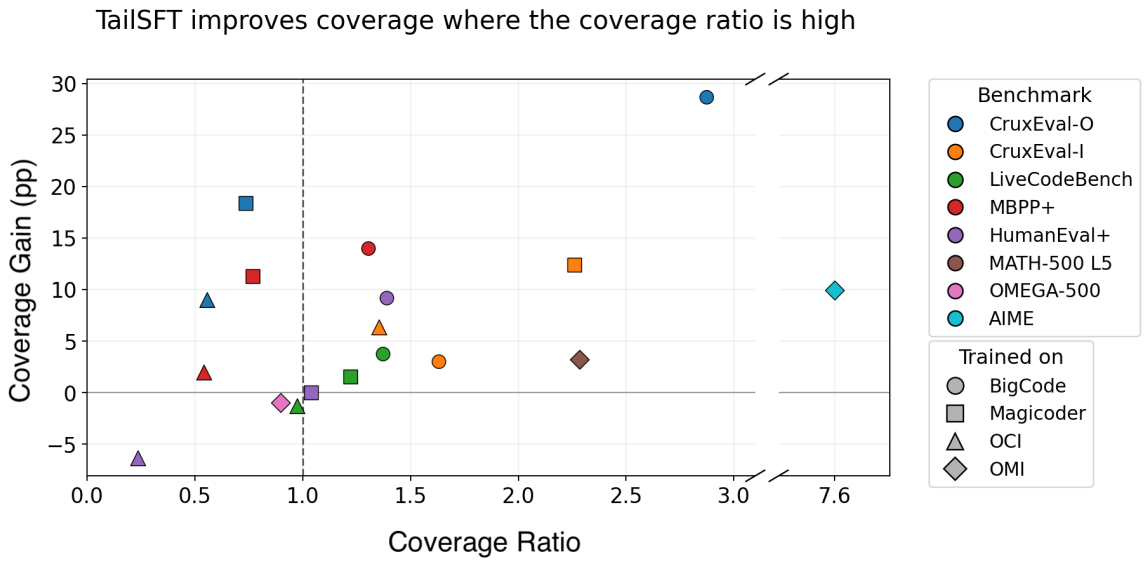

Figure 4: **When standard SFT loses more coverage than it gains, TailSFT preserves or improves coverage.** The vertical line marks $\rho_{16} = 1$, where the estimated coverage lost and gained by standard SFT are equal. Of the 11 evaluated settings with $\rho_{16} > 1$, 10 have positive coverage gains from TailSFT and one is essentially unchanged. Several settings with $\rho_{16} < 1$ also improve. Coverage gain and ratio are computed on the base-reachable set (Definition 4.1).

language model setting exhibits a measurable loss of coverage relative to the base model, and whether this coverage loss is predictive of the benefit of TailSFT.

We measure how standard SFT changes coverage on problems that are already reachable from the base model. Restricting attention to problems that are neither effectively unsolved nor already saturated under the base model, we use pass@1 to estimate the pass@16 that would result from independent sampling. We then aggregate the decreases and increases induced by standard SFT—their ratio measures whether standard SFT has lost more base-reachable coverage than it has gained.

Definition 4.1 (Coverage ratio)**.** *For problem i, model π, and $K \in \{1, 16\}$, let $P_{i,K}(\pi)$ denote the empirical pass@K, averaged over seeds. We convert pass@1 to an estimated pass@16 value using $f_{16}(p) := 1 - (1-p)^{16}$. The base-reachable set is $\mathcal{R}_0 := \{i : 0.05 < P_{i,16}(\pi_0) < 0.95\}$. The coverage lost and gained by standard SFT on this set are*

$$L := \sum_{i \in \mathcal{R}_0} \Big[f_{16}\big(P_{i,1}(\pi_0)\big) - f_{16}\big(P_{i,1}(\pi_{\mathrm{SFT}})\big) \Big]_+,$$

$$G := \sum_{i \in \mathcal{R}_0} \Big[f_{16}\big(P_{i,1}(\pi_{\mathrm{SFT}})\big) - f_{16}\big(P_{i,1}(\pi_0)\big) \Big]_+.$$

The coverage ratio is $\rho_{16} := L/G$. If $G = 0$, we set $\rho_{16} := \infty$.

Computing ρ_{16} requires only the base model and a single standard SFT run. A value $\rho_{16} > 1$ means that, on the base-reachable set, the estimated coverage lost during standard SFT exceeds the estimated coverage gained. This is the setting most directly targeted by TailSFT, which reduces training on responses that have already improved and thereby limits probability shifting away from responses represented by the base model.

The coverage ratio identifies a regime where TailSFT consistently preserves or improves coverage. In Figure 4, all 11 dataset–benchmark pairs satisfying $\rho_{16} > 1$ have nonnegative coverage gains (improvement in pass@16 on the base-reachable set $\mathcal{R}_0$) from TailSFT. Ten improve and one is essentially unchanged, with gains reaching an absolute 28.69% for BigCode evaluated on CruxEval-O. The condition is not necessary for improvement: several settings with $\rho_{16} < 1$ also benefit from TailSFT. Thus, the coverage ratio provides a conservative way to identify settings in which TailSFT preserves or improves coverage. Appendix D.3 contains further details.

		pass@1			pass@16		
SFT data	Benchmark	Standard	TailSFT	Δ	Standard	TailSFT	Δ
GRPO on MATH							
OMI	MATH-500 Level 5 (n=134)	$57.70_{\pm 0.44}$	$60.26_{\pm 1.02}$	+2.56	$83.83_{\pm 1.72}$	$87.06_{\pm 0.86}$	+3.23
	AIME 2022–2025 (n=120)	$14.40_{\pm 0.26}$	$15.61_{\pm 0.44}$	+1.21	$32.76_{\pm 0.88}$	$36.06_{\pm 0.98}$	+3.30
GRPO on MBPP+							
BigCode	MBPP+ (n=378)	$69.57_{\pm 0.29}$	$73.50_{\pm 0.70}$	+3.93	$75.93_{\pm 0.00}$	$78.66_{\pm 0.15}$	+2.73
Magicoder	MBPP+ (n=378)	$70.52_{\pm 0.25}$	$73.24_{\pm 0.09}$	+2.72	$78.22_{\pm 0.61}$	$80.60_{\pm 0.55}$	+2.38
OCI	MBPP+ (n=378)	$74.67_{\pm 0.08}$	$76.30_{\pm 0.05}$	+1.62	$84.22_{\pm 0.40}$	$84.04_{\pm 0.15}$	−0.18

Table 2: **Initializing from the TailSFT model improves post-GRPO performance across math and code.** GRPO is initialized from the Standard SFT and TailSFT checkpoints evaluated in Table 1, and rows are grouped by the corpus used for GRPO training: the MATH train split, excluding MATH-500, for the math rows, and the MBPP+ train split for the code rows. The SFT data column indicates the corpus used to train the initialization for each GRPO run. Values are percentages reported as mean ± standard deviation over three seeds, and Δ is improvement from TailSFT. Experiment details are in Appendix D.2.

4.3 GRPO Results

The coverage principle predicts that a higher-coverage initialization should expose RL to rewarding responses more often (Chen et al., 2026a). This section therefore tests whether the additional coverage resulting from TailSFT translates into better post-RL performance.

Setup. We initialize GRPO from the standard SFT and TailSFT checkpoints evaluated in Table 1. For math reasoning, we train on the MATH training split with MATH-500 held out and evaluate on MATH-500 Level 5 and AIME. For code, we train on the MBPP+ training split and evaluate on its test split. The GRPO procedure and hyperparameters are held fixed within each comparison. The main comparisons use 4 rollouts per prompt and an actor learning rate of 2×10^{-5}. Evaluation follows the OLMES protocol over three seed sets.

Improved coverage yields better post-RL performance. Initializing from TailSFT improves post-RL pass@1 in every matched comparison in Table 2, with absolute improvements ranging between 1.21% and 3.93%. Our coding experiments make the role of initialization especially salient—each TailSFT checkpoint has lower pass@1 than its standard SFT counterpart before RL, but has higher pass@1 afterward. Broader initial coverage indeed results in higher single-sample accuracy after GRPO. These results are consistent with the relationship between large-K performance and post-RL outcomes observed by Kang et al. (2026).

Improved coverage drives faster learning. Figure 8 shows that the difference between the two initializations appears early in training, with TailSFT runs beginning with lower reward but the gap quickly closing. In some settings, early reward increases up to 2.5× faster than for the corresponding standard SFT run. After RL, TailSFT retains higher pass@16 in four of the five matched comparisons, and is approximately tied in the fifth. The coverage preserved during SFT is therefore available to RL early in training and ultimately translates into higher pass@1.

5 Discussion

We present TailSFT as a drop-in replacement for standard SFT that filters already-fit sequences to improve coverage after the SFT stage, which produces improvements in post-RL performance. TailSFT is derived using a combination of theoretical analysis and controlled empirics, and significantly outperforms standard SFT in our language modeling experiments. More broadly, TailSFT uses the coverage principle to target the interaction between the SFT and RL stages of language model training (Chen et al., 2026a). We believe our results motivate a shift in how language models should be optimized, toward targeting the full training trajectory rather than fitting each stage independently.

References

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:2504.04030, 2025.

Rachit Bansal, Clara Mohri, Tian Qin, David Alvarez-Melis, and Sham M. Kakade. RL excursions during pre-training: How early is too early for on-policy learning? In *Workshop on Scaling Post-Training for LLMs*, 2026.

David Brandfonbrener, Hanlin Zhang, Andreas Kirsch, Jonathan Richard Schwarz, and Sham Kakade. CoLoR-Filter: Conditional loss reduction filtering for targeted language model pre-training. In *Advances in Neural Information Processing Systems*, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. arXiv:2407.21787, 2024.

Fan Chen, Audrey Huang, Noah Golowich, Sadhika Malladi, Adam Block, Jordan Ash, Akshay Krishnamurthy, and Dylan Foster. The coverage principle: How pre-training enables post-training. In *International Conference on Learning Representations*, 2026a.

Feng Chen, Allan Raventós, Nan Cheng, Surya Ganguli, and Shaul Druckmann. Rethinking fine-tuning when scaling test-time compute: Limiting confidence improves mathematical reasoning. In *Advances in Neural Information Processing Systems*, 2025.

Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *International Conference on Machine Learning*, 2019.

Yijie Chen, Yijin Liu, and Fandong Meng. SED-SFT: Selectively encouraging diversity in supervised fine-tuning. In *Annual Meeting of the Association for Computational Linguistics*, 2026b.

Zhoujun Cheng, Yutao Xie, Yuxiao Qu, Amrith Setlur, Shibo Hao, Varad Pimpalkhute, Tongtong Liang, Feng Yao, Zhengzhong Liu, Eric P. Xing, Virginia Smith, Ruslan Salakhutdinov, Zhiting Hu, Taylor W. Killian, and Aviral Kumar. IsoCompute playbook: Optimally scaling sampling compute for LLM RL. In *International Conference on Machine Learning*, 2026.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Aviral Kumar, Rishabh Agarwal, Sridhar Thiagarajan, Craig Boutilier, and Aleksandra Faust. Inference-aware fine-tuning for best-of-n sampling in large language models. In *International Conference on Learning Representations*, 2025.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *International Conference on Machine Learning*, 2025.

Zhiyuan Fan, Guanqiao Chen, Yanyi Huang, Mingkuan Zhao, Dadi Guo, and Yi R. Fung. Learning diverse responses with prefix-conditioned supervised fine-tuning. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Amir-massoud Farahmand, Csaba Szepesvári, and Rémi Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, 2010.

Dylan J. Foster, Akshay Krishnamurthy, David Simchi-Levi, and Yunzong Xu. Offline reinforcement learning: Fundamental barriers for value function approximation. In *Conference on Learning Theory*, 2022.

Dylan J. Foster, Zakaria Mhammedi, and Dhruv Rohatgi. Is a good foundation necessary for efficient reinforcement learning? the computational role of the base model in exploration. In *Conference on Learning Theory*, 2025.

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*, 2026.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A standard for language model evaluations. In *Findings of the Association for Computational Linguistics*, 2025.

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*, 2025.

Andre Wang He, Daniel Fried, and Sean Welleck. Rewarding the unlikely: Lifting GRPO beyond distribution sharpening. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Audrey Huang, Adam Block, Dylan Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *International Conference on Learning Representations*, 2025a.

Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J. Foster. Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment. In *International Conference on Machine Learning*, 2025b.

Zeyu Huang, Tianhao Cheng, Zihan Qiu, Zili Wang, Yinghui Xu, Edoardo M. Ponti, and Ivan Titov. Blending supervised and reinforcement fine-tuning with prefix sampling. In *International Conference on Machine Learning*, 2026.

Nan Jiang and Tengyang Xie. Offline reinforcement learning in large state spaces: Algorithms and guarantees. *Statistical Science*, 2025.

Hangzhan Jin, Sitao Luan, Sicheng Lyu, Guillaume Rabusseau, Doina Precup, and Mohammad Hamdaqa. RL fine-tuning heals the OOD forgetting in SFT. In *Workshop on Foundations of Reasoning in Language Models*, 2025.

Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline RL? In *International Conference on Machine Learning*, 2021.

Feiyang Kang, Michael Kuchnik, Karthik Padthe, Marin Vlastelica, Ruoxi Jia, Carole-Jean Wu, and Newsha Ardalani. Quagmires in SFT-RL post-training: When high SFT scores mislead and what to use instead. In *International Conference on Learning Representations*, 2026.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *International Conference on Learning Representations*, 2026.

Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning*, 2018.

Ziniu Li, Congliang Chen, Tian Xu, Zeyu Qin, Jiancong Xiao, Zhi-Quan Luo, and Ruoyu Sun. Preserving diversity in supervised fine-tuning of large language models. In *International Conference on Learning Representations*, 2025.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. Not all tokens are what you need for pretraining. In *Advances in Neural Information Processing Systems*, 2024.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning*, 2023.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *Advances in Neural Information Processing Systems*, 2025.

Nicholas Lourie, Michael Y. Hu, and Kyunghyun Cho. Scaling laws are unreliable for downstream tasks: A reality check. In *Findings of the Association for Computational Linguistics*, 2025.

Microsoft AI Team. Mai-thinking-1: Building a hill-climbing machine. Technical report, Technical report, Microsoft AI, 2026. https://microsoft. ai/pdf/mai-thinking . . . , 2025.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, 2022.

Xueyan Niu, Bo Bai, Wei Han, and Weixi Zhang. On the non-decoupling of supervised fine-tuning and reinforcement learning in post-training. arXiv:2601.07389, 2026.

Jinlong Pang, Na Di, Zhaowei Zhu, Jiaheng Wei, Hao Cheng, Chen Qian, and Yang Liu. Token cleaning: Fine-grained data selection for LLM supervised fine-tuning. In *International Conference on Machine Learning*, 2025.

Chongli Qin and Jost Tobias Springenberg. Supervised fine tuning on curated data is reinforcement learning (and can be improved). arXiv:2507.12856, 2025.

Ziheng Qin, Kai Wang, Zangwei Zheng, Jianyang Gu, Xiangyu Peng, Zhaopan Xu, Daquan Zhou, Lei Shang, Baigui Sun, Xuansong Xie, and Yang You. InfoBatch: Lossless training speed up by unbiased dynamic data pruning. In *International Conference on Learning Representations*, 2024.

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. arXiv:2402.03300, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained language models are harder to fine-tune. In *International Conference on Machine Learning*, 2025.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A. Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Conference on Empirical Methods in Natural Language Processing*, 2020.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun,

Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

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:2512.13961, 2026.

Raghuveer Thirukovalluru, Nicholas Monath, Bhuwan Dhingra, and Sam Wiseman. Sequence reducible holdout loss for language model pretraining. In *Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, 2024.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating AI for math with massive open-source instruction data. In *Workshop on Mathematical Reasoning and AI*, 2024.

Jens Tuyls, Dylan Foster, Akshay Krishnamurthy, and Jordan Ash. Representation-based exploration for language models: From test-time to post-training. In *International Conference on Learning Representations*, 2026.

Hao Wang, Hao Gu, Hongming Piao, Kaixiong Gong, Yuxiao Ye, Xiangyu Yue, Sirui Han, Yike Guo, and Dapeng Wu. Learning while staying curious: Entropy-preserving supervised fine-tuning via adaptive self-distillation for large reasoning models. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Jiachen T. Wang, Tong Wu, Dawn Song, Prateek Mittal, and Ruoxi Jia. GREATS: Online selection of high-quality data for LLM training in every iteration. In *Advances in Neural Information Processing Systems*, 2024.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. OctoThinker: Mid-training incentivizes reinforcement learning scaling. In *AI for Math Workshop*, 2025.

Ishaan Watts, Catherine Li, Sachin Goyal, Jacob Mitchell Springer, and Aditi Raghunathan. Sharpness-aware pretraining mitigates catastrophic forgetting. In *International Conference on Machine Learning*, 2026.

Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and LINGMING ZHANG. Selfcodealign: Self-alignment for code generation. In *Advances in Neural Information Processing Systems*, 2024a.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and LINGMING ZHANG. Magicoder: Empowering code generation with OSS-instruct. In *International Conference on Machine Learning*, 2024b.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. In *International Conference on Learning Representations*, 2026.

Chen Wu, Sachin Goyal, and Aditi Raghunathan. Mode-conditioning unlocks superior test-time compute scaling. In *International Conference on Learning Representations*, 2026.

Fang Wu and Yejin Choi. The invisible leash: Why RLVR may not escape its origin. In *AI for Math Workshop*, 2025.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning*, 2024.

Tengyang Xie and Nan Jiang. Q* approximation schemes for batch reinforcement learning: A theoretical comparison. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

Tengyang Xie, Dylan J. Foster, Yu Bai, Nan Jiang, and Sham M. Kakade. The role of coverage in online reinforcement learning. In *International Conference on Learning Representations*, 2023.

Kazuki Yano, Shun Kiyono, Sosuke Kobayashi, Sho Takase, and Jun Suzuki. Pre-training LLM without learning rate decay enhances supervised fine-tuning. In *International Conference on Learning Representations*, 2026.

Jian Yao, Ran Cheng, Xingyu Wu, Jibin Wu, and KC Tan. Diversity-aware policy optimization for large language model reasoning. In *Advances in Neural Information Processing Systems*, 2025.

Hiroshi Yoshihara, Taiki Yamaguchi, and Yuichi Inoue. A practical two-stage recipe for mathematical LLMs: Maximizing accuracy with SFT and efficiency with reinforcement learning. In *AI for Math Workshop*, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025.

Hansi Zeng, Kai Hui, Honglei Zhuang, Zhen Qin, Zhenrui Yue, Hamed Zamani, and Dana Alon. Can pre-training indicators reliably predict fine-tuning outcomes of LLMs? arXiv:2504.12491, 2025.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. In *International Conference on Machine Learning*, 2026a.

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*, 2026b.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026a.

Feng Zhao, Hong Zhang, Yu Yang, Ruilin Zhao, and Guandong Xu. Don't force the fit: Bounded log-likelihood loss for enhanced reasoning in large language models. In *International Conference on Machine Learning*, 2026b.

Rosie Zhao, Alexandru Meterez, Sham M. Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. Echo chamber: RL post-training amplifies behaviors learned in pretraining. In *Conference on Language Modeling*, 2025.

A Additional Related Work

Data selection and diversity-preserving SFT. Importance sampling and dynamic pruning prioritize high-gradient or low-information examples to accelerate optimization while preserving the full-data objective (Katharopoulos and Fleuret, 2018; Qin et al., 2024); TailSFT instead changes the effective SFT objective to improve coverage and post-RL performance. RHO-LOSS selects learnable, not-yet-learned examples and Dataset Cartography diagnoses examples from training dynamics (Mindermann et al., 2022; Swayamdipta et al., 2020); TailSFT uses only loss reduction relative to the initial policy and masks the most-improved sequences online. Rho-1 and CoLoR use reference-relative losses in pre-training, Token Cleaning selects SFT tokens, and GREATS and LESS estimate utility or influence (Lin et al., 2024; Brandfonbrener et al., 2024; Pang et al., 2025; Wang et al., 2024; Xia et al., 2024); TailSFT operates at sequence or document granularity without target examples or influence computation. Qin and Springenberg (2025) cast curated SFT as implicit RL; TailSFT supplies a dynamic, coverage-motivated rule and evaluates an explicit later RL stage. Confidence caps, entropy regularization, self-distillation, and token-level clipping preserve diversity during SFT (Chen et al., 2025; Li et al., 2025; Zhao et al., 2026b; Wang et al., 2026; Chen et al., 2026b), while prefix-conditioned SFT separates response modes (Fan et al., 2026); TailSFT instead masks already-improved examples relative to the base policy, without auxiliary losses or mode labels.

Coupling supervised and reinforcement training. Bansal et al. (2026) apply RL at intermediate pre-training checkpoints and blend SFT and RL updates, Fu et al. (2026) jointly weight demonstrations and on-policy rollouts, Huang et al. (2026) continue expert prefixes on-policy, and Zhang et al. (2026b) retain expert SFT as a dynamically weighted RL auxiliary objective. These methods change when or how the objectives are coupled; TailSFT leaves the two-stage pipeline and RL algorithm unchanged and changes only which SFT examples receive gradient. Yoshihara et al. (2025) use prolonged SFT to maximize mathematical accuracy and then GRPO chiefly to improve token efficiency; TailSFT instead optimizes the SFT stage for the final post-RL model, even when local SFT metrics worsen. Niu et al. (2026) derive interference between sequential SFT and RL objectives under their assumptions; TailSFT keeps the stages sequential but reshapes SFT gradients to better support the later RL objective.

Coverage and test-time compute. The coverage principle formalizes response-level likelihood-ratio coverage as the condition governing Best-of-N recovery (Chen et al., 2026a); TailSFT operationalizes this criterion as an online SFT rule and proves an advantage for relative clipping. Classical RL theory uses concentrability and related coverage coefficients to control error propagation and offline or online learnability (Farahmand et al., 2010; Chen and Jiang, 2019; Xie and Jiang, 2020; Jin et al., 2021; Foster et al., 2022; Jiang and Xie, 2025; Xie et al., 2023); those works ask whether logged state-action data cover a target policy, whereas TailSFT asks whether a language model covers expert responses at a finite sampling budget. Repeated sampling yields predictable coverage gains (Brown et al., 2024), while compute allocation, Best-of-N analysis, inference-aware fine-tuning, mode conditioning, and rollout-budget scaling improve how a fixed policy is sampled or trained for sampling (Snell et al., 2025; Huang et al., 2025b; Chow et al., 2025; Wu et al., 2026; Cheng et al., 2026); TailSFT instead improves the pre-RL policy without a verifier, test-time mode label, or altered rollout allocation.

RLVR, sharpening, and exploration. Sharpening theory formalizes post-training as reallocating mass within covered behaviors, while sampling analyses show that capabilities can already be latent in the base policy (Huang et al., 2025a; Karan and Du, 2026); TailSFT builds on this view by improving useful coverage before RL. Empirically, RLVR can raise pass@1 without expanding large-k capacity, remain support-constrained, or amplify earlier behaviors (Yue et al., 2025; Wu and Choi, 2025; Zhao et al., 2025), while Wen et al. (2026) show that verifiable rewards can promote correct reasoning latent in the base model; TailSFT changes SFT so more reward-bearing responses remain sampleable. RL-stage methods upweight unlikely correct trajectories, optimize solution diversity, or add representation-based exploration (He et al., 2025; Yao et al., 2025; Tuyls et al., 2026); these methods are complementary, whereas TailSFT uses supervised data alone. Liu et al. (2025) show that prolonged, diverse RL can expand the reasoning boundary, and Foster et al. (2025) show that base-policy coverage controls efficient exploration; TailSFT does not posit an absolute boundary, but supplies a better foundation for standard on-policy RL.

Stage-aware model development. Kang et al. (2026) show that high SFT scores need not predict post-RL performance and identify held-out generalization loss and pass@large-k as stronger proxies; TailSFT goes beyond diagnosis by modifying SFT to improve coverage and validating the resulting initialization through matched RL runs. Models with the same pre-training loss can transfer differently because of optimization's implicit bias (Liu et al., 2023); pre-training perplexity can misrank fine-tuning outcomes (Zeng et al., 2025), and downstream scaling laws can be unstable (Lourie et al., 2025). These works expose failures of local metrics; TailSFT supplies an actionable SFT intervention and a coverage-based diagnostic for the SFT-to-RL transition. Extended pre-training can reduce loss while harming adaptability, alternative schedules can improve SFT despite weaker local metrics, and flatter pre-training solutions can reduce forgetting after post-training (Springer et al., 2025; Yano et al., 2026; Watts et al., 2026); TailSFT makes the analogous intervention inside SFT and targets response coverage rather than pre-training geometry. Zhang et al. (2026a) and Wang et al. (2025) show that pre- or mid-training exposure controls later RL gains; TailSFT isolates a lightweight SFT intervention and tests it through matched RL runs. Chu et al. (2025) separate SFT's stabilizing role from RL's generalization, while Jin et al. (2025) find that OOD performance can peak early in SFT and be partly restored by RL; TailSFT filters already-fit examples to prevent coverage loss before RL.

B Theoretical Analysis: TailSFT in the expert conditioning setting

We consider a stylized setup for supervised fine-tuning called *expert conditioning* where the SFT data distribution and optimal policy for the downstream task are defined by conditioning the pretrained model output distribution on some event. For theoretical analysis, we focus on a simplified setting where there is no context or prompt, and so all policies are elementary distributions over responses $y \in \mathcal{Y}$. Let $\pi_{\text{ref}} \in \Delta(\mathcal{Y})$ be the pretrained model and let $S \subset \mathcal{Y}$ be some subset of responses. In expert conditioning, we define the expert policy as $\pi^\star(y) \propto \pi_{\text{ref}}(y) \cdot \mathbf{1}\{y \in S\}$. Note that this setup is closely related to formulations of RLHF where the expert is defined as $\pi_{\text{ref}}(y) \cdot \exp(R^\star(y)/\beta)$ for some reward function $R^\star$; formally, expert conditioning is equivalent to this formulation in the limit where $\beta \to 0$.

We are given n samples $y_1, \ldots, y_n \sim \pi^\star$ and obtain a policy by solving a certain optimization problem defined in terms of a loss function over the sample. The three loss functions are

$$\begin{aligned}
\text{ERM}: \qquad & L_{\text{ERM}}(\pi) := \frac{1}{n}\sum_{i=1}^{n} -\log(\pi(y_i)) \\
\text{ABS}: \qquad & L_{\text{ABS},\alpha}(\pi) := \frac{1}{n}\sum_{i=1}^{n} \max(-\log(\pi(y_i)) + \log(\alpha), 0) \\
\text{OFF}: \qquad & L_{\text{OFF},\beta}(\pi) := \frac{1}{n}\sum_{i=1}^{n} \max(-\log(\pi(y_i)) + \log(\beta \cdot \pi_{\text{ref}}(y_i)), 0)
\end{aligned}$$

The first objective is standard empirical risk minimization on the cross entropy loss. The second and third are TailSFT variants with absolute and relative clipping respectively. Indeed for absolute clipping, we ignore sample y_i if $\pi(y_i) \geq \alpha$ and for relative clipping we ignore y_i if $\pi(y_i) \geq \beta\pi_{\text{ref}}(y_i)$. The formal optimization problem for a particular loss function is:

$$\underset{\pi \in \Delta(\mathcal{Y})}{\text{minimize}} \ \text{KL}(\pi \| \pi_{\text{ref}}) \ \text{ subject to } \ L(\pi) = \min_{\pi'} L(\pi') \tag{1}$$

Thus we minimize the forward Kullback-Leibler (KL) divergence between the optimization variable π and the reference policy π_{ref} subject to π being among the minimizers of the loss L. We use forward KL as simplification of the implicit bias of optimization, and use forward KL primarily due to its mode-covering behavior, so that coverage is preserved to the extent possible subject to minimizing the loss. We take L to be $L_{\text{ERM}}, L_{\text{ABS},\alpha}, L_{\text{OFF},\beta}$ accordingly.

Formally, fixing the dataset, define $\pi_{\text{ERM}}, \pi_{\text{ABS},\alpha}, \pi_{\text{OFF},\beta}$ to be the optimizers of Eq. (1) with loss functions $L_{\text{ERM}}, L_{\text{ABS},\alpha}, L_{\text{OFF},\beta}$ respectively. We are interested in qualitatively understanding how well these optimizers

cover the expert policy $\pi^\star$, where recall that we define coverage as:

$$\mathrm{Cov}_N(\pi) := \Pr_{y\sim\pi^\star}\left[\frac{\pi^\star(y)}{\pi(y)} \geq N\right]$$

Our main result is as follows:

Theorem B.1. *We have the following comparisons:*

1. ***Offset clipping is always preferred**: For all π_{ref}, S and datasets:*

$$\inf_\beta \mathrm{Cov}_N(\pi_{\mathrm{OFF},\beta}) \leq \inf_{\beta:\min_\pi L_{\mathrm{OFF},\beta}(\pi)=0} \mathrm{Cov}_N(\pi_{\mathrm{OFF},\beta}) \leq \left(\inf_\alpha \mathrm{Cov}_N(\pi_{\mathrm{ABS},\alpha})\right) \wedge \mathrm{Cov}_N(\pi_{\mathrm{ERM}})$$

2. ***Offset clipping can dominate**: For $|\mathcal{Y}|$ sufficiently large, there exists a reference policy π_{ref} and expert subset S such that with probability at least $1-\mathrm{poly}(|\mathcal{Y}|^{-1})$*

$$\inf_\beta \mathrm{Cov}_N(\pi_{\mathrm{OFF},\beta}) < \left(\inf_\alpha \mathrm{Cov}_N(\pi_{\mathrm{ABS},\alpha})\right) \wedge \mathrm{Cov}_N(\pi_{\mathrm{ERM}})$$

3. ***Absolute clipping can be worse than ERM**: There exists a reference policy π_{ref} and expert subset S such that with high probability*

$$\mathrm{Cov}_N(\pi_{\mathrm{ERM}}) < \inf_{\alpha:\min_\pi L_{\mathrm{ABS},\alpha}(\pi)=0} \mathrm{Cov}_N(\pi_{\mathrm{ABS},\alpha})$$

   That is, in the regime where zero loss is achievable for absolute clipping, we can have that empirical risk minimization strictly dominates absolute loss clipping.

Note that the first claim implies that offset clipping—even when restricted to the parameter regime where zero loss is achievable–is never worse than empirical risk minimization.

B.1 Proof of Theorem B.1

First we derive a structural characterization of the solutions of Eq. (1). Next, we use this characterization to understand the coverage of the policies $\pi_{\mathrm{ERM}}, \pi_{\mathrm{ABS},\alpha}, \pi_{\mathrm{OFF},\beta}$. Finally we establish the comparisons in the theorem statement.

Lemma B.1 (Structure of KL projections)**.** *Let $\{f_y : y \in \mathcal{Y}\} \subset [0,1]$ satisfy $\sum_y f_y \leq 1$. Consider the optimization problem*

$$\widehat{\pi} \leftarrow \arg\min_{\pi\in\Delta(\mathcal{Y})} \mathrm{KL}(\pi\|\|\pi_{\mathrm{ref}}) \text{ subject to } \forall y : \pi(y) \geq f_y.$$

Then the minimizer $\widehat{\pi}$ is unique and given by

$$\widehat{\pi}(y) = \max(f_y, \lambda\pi_{\mathrm{ref}}(y)) \quad \text{where} \quad \sum_y \max(f_y, \lambda\pi_{\mathrm{ref}}(y)) = 1.$$

Proof of Lemma B.1. Observe that the optimization problem is strictly convex and feasible under the condition that $\sum_y f_y \leq 1$. We write the Lagrangian of the optimization problem as

$$\sum_y \pi(y)\log\frac{\pi(y)}{\pi_{\mathrm{ref}}(y)} + \lambda\left(\sum_y \pi(y) - 1\right) + \sum_y \mu_y(f(y) - \pi(y)) \qquad \mu_y \geq 0.$$

The stationary conditions are

$$\forall y : \ \log\frac{\pi(y)}{\pi_{\mathrm{ref}}(y)} + 1 + \lambda - \mu_y = 0.$$

If $\pi(y) > f_y$, so the constraint is inactive, then by complementary slackness we have $\mu_y = 0$, so $\pi(y) = \pi_{\mathrm{ref}}(y) \cdot \exp(-1-\lambda)$ and this must be larger than f_y. Otherwise, we must have $\pi(y) = f_y$ solving the stationary condition for μ_y gives

$$\mu_y = \log(\pi(y)/\pi_{\mathrm{ref}}(y)) + 1 + \lambda = \log \frac{f_y}{\pi_{\mathrm{ref}}(y) \cdot \exp(-1-\lambda)}$$

The constraint that $\mu_y \geq 0$ thus implies that $f_y \geq \pi_{\mathrm{ref}}(y) \cdot \exp(-1-\lambda)$. Taken together, this gives $\pi(y) = \max(f_y, \pi_{\mathrm{ref}}(y)\exp(-1-\lambda))$ and λ is chosen to normalize the distribution. □

Lemma B.2 (Structure of clipped minimizer)**.** *Let $\{f_y : y \in \mathcal{Y}\}$ let $p \in \Delta(Y)$ and let $\sum_y \mathbf{1}\{p(y) > 0\} f_y > 1$. Then the minimizer of*

$$\sum_{y \in \mathcal{Y}} p(y) \max(-\log \pi(y) + \log(f_y), 0)$$

is unique and given by

$$\hat{\pi}(y) = \min(f_y, \lambda p(y)) \quad \text{where} \quad \sum_y \min(f_y, \lambda p(y)) = 1.$$

Proof. First observe that for any y such that $p(y) = 0$ we will have $\pi(y) = 0$ as well, because the conditions on f_y imply that we cannot achieve an objective value of zero, and we can always shift mass from actions with $p(y) = 0$ to those with $p(y) \neq 0$ to reduce the loss. Thus, without loss of generality we can assume that $p(y) \neq 0$ which also implies $\pi(y) \neq 0$. Next, we write the Lagrangian:

$$\sum_y p(y) \max(-\log \pi(y) + \log(f_y), 0) + \lambda \left(\sum_y \pi(y) - 1 \right)$$

The stationary condition is that $\frac{-p(y)}{\pi(y)} + \lambda = 0$ if $\pi(y) < f_y$. If $\pi(y) = f_y$) the sub-differential for the loss term is in $[-p(y)/\pi(y), 0]$ and if $\pi(y) > 0$ we must have $\lambda = 0$ to satisfy stationarity. However, since at least one y must have $\pi(y) < f_y$ we get that $\lambda > 0$. Therefore no action can have $\pi(y) > f_y$ and we have that $\pi(y) = \min(f_y, \lambda p(y))$. □

Next we characterize the structure and the coverage of the $\pi_{\mathrm{OFF},\beta}$.

Lemma B.3 (Coverage of $\pi_{\mathrm{OFF},\beta}$)**.** *For any dataset, let $\hat{S} \subseteq S$ be the actions observed in the dataset. We have*

$$\inf_{\beta \geq 0} \mathrm{Cov}_N(\pi_{\mathrm{OFF},\beta}) \leq \inf_{\beta : \min_\pi L_{\mathrm{OFF},\beta}(\pi) = 0} \mathrm{Cov}_N(\pi_{\mathrm{OFF},\beta}) = \begin{cases} 0, & N \geq 1/\pi_{\mathrm{ref}}(S) \\ \frac{\pi_{\mathrm{ref}}(S \setminus \hat{S})}{\pi_{\mathrm{ref}}(S)}, & 1 < N < 1/\pi_{\mathrm{ref}}(S) \end{cases}$$

Proof of Lemma B.3. If $N \geq 1/\pi_{\mathrm{ref}}(S)$ then take $\beta = 1$ so that π_{ref} itself is a minimizer of $L_{\mathrm{OFF},\beta}$, i.e., $L_{\mathrm{OFF},\beta}(\pi_{\mathrm{ref}}) = 0$. With this choice of β, clearly π_{ref} is the solution to Eq. (1). To compute the coverage, observe that for every action y, $\pi^\star(y)/\pi_{\mathrm{ref}}(y) = 1/\pi_{\mathrm{ref}}(S) \leq N$. Thus, in this case $\mathrm{Cov}(\pi_{\mathrm{ref}}) = 0$ and with optimally tuned β, the KL projection onto the set of offset-loss minimizers preserves this coverage.

If $1 < N < 1/\pi_{\mathrm{ref}}(S)$ we first translate the offset loss optimization problem into the form in Lemma B.1 and then compute the normalizing constant λ in the distribution. Observe that the offset loss only involves observed actions $y \in \hat{S}$. If β is such that 0 offset loss is feasible, then we have

$$L_{\mathrm{OFF},\beta}(\pi) = 0 \Leftrightarrow \forall y \in \hat{S} : \ \pi(y) \geq \beta \pi_{\mathrm{ref}}(y) \text{ and } \forall y \notin \hat{S} : \pi(y) \geq 0$$

Here the first condition arises from achieving zero loss, while the second condition arises because π must be a distribution. Thus the offset loss version of Eq. (1) corresponds to taking $f_y = \beta \pi_{\mathrm{ref}}(y)$ if $y \in \hat{S}$ and $f_y = 0$ otherwise in Lemma B.1. Next, observing that every action in the minimizing distribution has mass at least

$\lambda\pi_{\text{ref}}(y)$ we get that $\lambda \le 1$. Since $f_y = 0$ for actions $y \notin \hat{S}$, we have that $\pi_{\text{OFF},\beta}(y) = \lambda\pi_{\text{ref}}(y)$ for all $y \notin \hat{S}$. Such actions that are further supported by $\pi^\star$ are uncovered, since

$$\frac{\pi^\star(y)}{\pi_{\text{OFF},\beta}(y)} \ge \frac{\pi^\star(y)}{\pi_{\text{ref}}(y)} = \frac{1}{\pi_{\text{ref}}(S)} > N$$

The first inequality uses that $\lambda \le 1$ and the second inequality is by our assumed regime for N. Thus we obtain for any feasible β we obtain a coverage lower bound

$$\text{Cov}_N(\pi_{\text{OFF},\beta}) \ge \frac{\pi_{\text{ref}}(S \setminus \hat{S})}{\pi_{\text{ref}}(S)}.$$

To show that this is achievable take $\beta = (\pi_{\text{ref}}(S)N)^{-1} > 1$ and observe that

$$\beta\pi_{\text{ref}}(S) = 1/N < 1.$$

Thus if we set $\pi(y) = \beta\pi_{\text{ref}}(y)$ for $y \in S$ we achieve 0 loss and have $\sum_y \pi(y) < 1$; we can allocate the remaining mass arbitrarily to preserve 0 offset loss and obtain a distribution.

Since zero-loss is achievable, the KL projection ensures that $\pi_{\text{OFF},\beta}(y) \ge \beta\pi_{\text{ref}}(y)$ for all $y \in \hat{S}$ as this is required to achieve zero loss. By the choice of β this actions are covered. On the other hand, all other actions have $\pi_{\text{OFF},\beta}(y) = \lambda\pi_{\text{ref}}(y) \le \pi_{\text{ref}}(y)$ using the argument above that $\lambda \le 1$. These actions are all not covered under the condition that $1/\pi_{\text{ref}}(S) > N$. Thus this choice of β achieves coverage exactly $\pi_{\text{ref}}(S \setminus \hat{S})/\pi_{\text{ref}}(S)$. □

Proof of Claim 1 of Theorem B.1. We first argue about π_{ERM}. Note that the ERM is unique and is exactly the empirical distribution over the sample. In particular, we have $\pi_{\text{ERM}}(S \setminus \hat{S}) = 0$, and therefore these actions are always uncovered, so $\text{Cov}_N(\pi_{\text{ERM}}) \ge \pi_{\text{ref}}(S \setminus \hat{S})/\pi_{\text{ref}}(S)$.

Next we turn to $\pi_{\text{ABS},\alpha}$. Here we consider two cases. First consider the case that α is large enough such that zero loss is not achievable. In this case, if we have a policy π that allocates mass to unobserved actions, we can always strictly improve the loss by moving that mass onto some observed action that is not already saturated, and such an action must exist if zero loss is not achievable. Thus the loss minimizer must allocate no mass on unobserved actions. In this case the same reasoning as we applied to analyze the ERM holds.

Next, consider that α is small enough such that zero loss is achievable. As with the offset loss, we can translate the optimization problem to the form in Lemma B.1 and take $f_y = \alpha\mathbf{1}\{y \in \hat{S}\}$. However, as in the proof of Lemma B.3, we still require that $\lambda \le 1$. As before when $1 < N < 1/\pi_{\text{ref}}(S)$, this implies that all unobserved actions in S are uncovered, establishing a lower bound of $\pi_{\text{ref}}(S \setminus \hat{S})/\pi_{\text{ref}}(S)$ in this regime. □

Proof of Claim 2 in Theorem B.1. We construct an instance with three types of arms with $|\mathcal{Y}| = 1+n/4+3n$ for some $n \in \mathbb{N}$ such that $\mathcal{Y} = \{o\} \cup H \cup L$ where $|H| = n/4$ and $L = 3n$. Define

$$\pi_{\text{ref}}(o) = \frac{1}{2}, \quad \pi_{\text{ref}}(h) = \frac{1}{n}, h \in H, \quad \pi_{\text{ref}}(\ell) = \frac{1}{12n}, \ell \in L.$$

Set $S = H \cup L$ such that

$$\pi^\star(h) = \frac{2}{n}, h \in H, \quad \pi^\star(\ell) = \frac{1}{6n}, \ell \in L$$

Recall that $\hat{S}$ is the set of actions observed in the sample. The high probability event is the intersection of two events: (1) $|\hat{S}|$ is sufficiently large and (2) $|\hat{H}^{(1)}|$ is sufficiently large, where $\hat{H}^{(1)}$ is the set of actions in H that are observed exactly once.

To control $|\hat{S}|$, we first calculate:

$$\mathbb{E}|\hat{S} \cap H| = \sum_{h \in H} \Pr[h \in \hat{S}] = \frac{n}{4}(1 - (1 - 2/n)^n) \ge \frac{n}{4}(1 - e^{-2})$$

$$\mathbb{E}|\hat{S} \cap L| = \sum_{\ell \in L} \Pr[\ell \in \hat{S}] = 3n(1 - (1 - 1/(6n))^n) \ge 3n(1 - e^{-1/6})$$

Observe that $|\hat{S}| = |\hat{S} \cap H| + |\hat{S} \cap L|$ and that the random variable $|\hat{S}|$ satisfies the conditions of McDiarmid's inequality with constant 1. Thus with probability at least $1 - \delta/2$ we have

$$|\hat{S}| \geq \mathbb{E}|\hat{S}| - \sqrt{\frac{n \log(2/\delta)}{2}} \geq \frac{n}{4}(1 - e^{-2}) + 3n(1 - e^{-1/6}) - \sqrt{\frac{n \log(2/\delta)}{2}}.$$

The constant on the $\Theta(n)$ term is at least 0.676, and for $\delta = \text{poly}(1/n)$ the $\Theta(n)$ term dominates, and so we have that for n sufficiently large, $|\hat{S}| \geq 2n/3$.

We control $|\hat{H}^{(1)}|$ similarly:

$$\mathbb{E}|\hat{H}^{(1)}| = \sum_{h \in H} \Pr[B_h = 1] = \frac{n}{4}\left(n \cdot \frac{2}{n} \cdot (1 - 2/n)^{n-1}\right) \geq \frac{n}{2} \cdot (1 - 2/n)^{n-1}$$

Here B_h is a Binomially distributed random variable with parameters $(n, 2/n)$. Again we apply McDiarmid's inequality, here $|\hat{H}^{(1)}|$ satisfies the conditions with a constant of 2, so that with probability at least $1 - \delta/2$ we have

$$|\hat{H}^{(1)}| \geq \mathbb{E}|\hat{H}^{(1)}| - \sqrt{2n \log(2/\delta)} \geq \frac{n}{2} \cdot (1 - 2/n)^{n-1} - \sqrt{2n \log(2/\delta)}$$

Observe that the first term asymptotically approaches $n \cdot e^{-2}/2$ while the second term is lower order in n. Therefore for n sufficiently large, we can see that $|\hat{H}^{(1)}| \geq e^{-2}/3$ with high probability.

Next we turn to the coverage calculations. Since $N = 4/3$ satisfies $1 < N < 1/\pi_{\text{ref}}(S) = 2$ we know that $\inf_{\beta \geq 0} \text{Cov}(\pi_{\text{OFF},\beta}) = \pi_{\text{ref}}(S \setminus \hat{S})/\pi_{\text{ref}}(S) = \pi^\star(S \setminus \hat{S})$. This is exactly the expert mass of the unobserved set. On the other hand π_{ERM} matches exactly the empirical frequencies in the data. In particular, we have $\pi_{\text{ERM}}(h) = 1/n$ for $h \in \hat{H}^{(1)}$. This implies

$$\text{Cov}(\pi_{\text{ERM}}) \geq \sum_{y \in S \setminus \hat{S}} \pi^\star(y) + \sum_{y \in \hat{H}^{(1)}} \frac{2}{n} \mathbf{1}\left\{\frac{2/n}{1/n} \geq N\right\} = \pi^\star(S \setminus \hat{S}) + \pi^\star(\hat{H}^{(1)})$$

Finally for $\pi_{\text{ABS},\alpha}$ we need to show that no absolute clipping threshold improves on this value. To cover any $h \in \hat{H}^{(1)}$ we need $\pi_{\text{ABS},\alpha}(h) \geq 3/(2n)$. Recall that we only see $|\hat{S}|$ actions in the dataset. If we set $\alpha \leq 1/|\hat{S}|$ then we can achieve zero loss but from Lemma B.1 we have

$$\pi_{\text{ABS},\alpha}(h) = \max(\alpha, \lambda \pi_{\text{ref}}(h)) \leq \max(1/|\hat{S}|, 1/n) < 3/(2n).$$

Here recall that $\lambda \leq 1$ (otherwise $\pi_{\text{ABS},\alpha}(h)$ will not be a distribution) and that we have $|\hat{S}| > \frac{2n}{3}$ with high probability. In this regime, unobserved actions get mass $\lambda \pi_{\text{ref}}(h)$ but this is insufficient to cover $\pi^\star$ (since $2 = \pi^\star(y)/\pi_{\text{ref}}(y) = 2 \geq N = 4/3$).

On the other hand, if $\alpha > 1/|\hat{S}|$ then, by Lemma B.2 the unique minimizer of the absolute clipped loss is

$$\pi_{\text{ABS},\alpha}(y) = \min(\alpha, \lambda \hat{\mu}(y))$$

where $\hat{\mu}(y)$ is the empirical distribution. The normalizing condition gives,

$$1 \geq \sum_{y \in \hat{S}} \min(\alpha, \lambda \hat{\mu}(y)) \geq |\hat{S}| \min(\alpha, \lambda/n),$$

which follows because every observed action is observed at least once, so it has empirical mass at least $1/n$. Since $\alpha > 1/|\hat{S}|$ we must have $\lambda/n \leq 1/|\hat{S}|$ and therefore actions $h \in \hat{H}^{(1)}$ have $\pi_{\text{ABS},\alpha}(h) \leq 1/|\hat{S}| \leq \frac{3}{2n}$ (while unobserved actions have $\pi_{\text{ABS},\alpha}(y) = 0$).

Thus in both cases we have

$$\text{Cov}(\pi_{\text{ABS},\alpha}) \geq \pi^\star(S \setminus \hat{S}) + \pi^\star(\hat{H}^{(1)})$$

The high probability event implies that $|\hat{H}^{(1)}| > 0$ and so $\pi^\star(\hat{H}^{(1)}) > 0$, establishing strict separation. □

Proof of Claim 3 in Theorem B.1. We use the same construction as in the proof of Claim 2. Recall that in that construction, zero absolute clipped loss is achievable only if $\alpha \le 1/|\hat{S}|$. Let $\hat{H}^{(>1)}$ denote the set of actions in H that are observed at least twice in the sample $\hat{S}$. Since $\alpha \le 1/|\hat{S}| \le 3/(2n)$ (since with high probability we have $|\hat{S}| \ge 2n/3$), via Lemma B.1, the absolute clipped loss solution satisfies

$$\pi_{\mathrm{ABS},\alpha}(h) = \max(\alpha, \lambda\pi_{\mathrm{ref}}(h)) \le \max(1/|\hat{S}|, 1/n) < 3/(2n) = \pi^\star(h)/N,$$

if we take $N = 4/3$ as above. Thus all actions in $\hat{H}^{(<1)}$ violate coverage. We already saw that all actions in $\hat{H}^{(1)}$ violate coverage as well as all unobserved actions in L. Thus the coverage is

$$\inf_{\alpha:\min_\pi L_{\mathrm{ABS},\alpha}(\pi)=0} \mathrm{Cov}(\pi_{\mathrm{ABS},\alpha}) = \frac{1}{2} + \pi^\star(L \setminus \hat{S}).$$

On the other hand, ERM covers actions in $\hat{H}^{(>1)}$, because for any $h \in \hat{H}^{(>1)}$, we have $\pi_{\mathrm{ERM}}(h) \ge 2/n$ while $\pi^\star(h) = 2/n$. Thus ERM strictly dominates absolute clipping whenever $|\hat{H}^{(>1)}| > 0$. Since

$$\mathbb{E}|\hat{H}^{(>1)}| = \sum_{h\in H} \Pr[B_h > 1] = \frac{n}{4} \cdot (1 - (1-2/n)^n - 2(1-2/n)^{n-1})$$

where $B_h \sim \mathrm{Binomial}(n, 2/n)$, and since $|\hat{H}^{(>1)}|$ satisfies the conditions of McDiarmid's inequality with constant 1, we have that with probability at least $1-\delta$

$$|\hat{H}^{(>1)}| \ge \frac{n}{4} \cdot (1 - (1-2/n)^n - 2(1-2/n)^{n-1}) - \sqrt{\frac{n}{2}\log(1/\delta)}.$$

For sufficiently large n the first term is larger than $(1-3e^{-2})n/5$ and the second term is lower order, so for sufficiently large n we get that $|H^{(>1)}| > 1$ with high probability. □

C Details for Graph Navigation Experiments

In this section, we describe the task, training details, and results for the graph navigation experiments presented in Figure 3. The setting is closely related to that of Chen et al. (2026a); we present all details for completeness but highlight where our experiments diverge from theirs.

C.1 Graph Reasoning Task and Data

Following Chen et al. (2026a), we use a path-following task in a directed acyclic graph (DAG) as an abstraction for reasoning problems. The setup builds on a long line of work using synthetic tasks to understand phenomena in language modeling and serves as a minimal, expressive, yet flexible setting to develop interventions like TailSFT.

The task is path-following a directed acyclic graph. Each prompt $x \in \mathcal{X}$ encodes a graph G along with a source node s and a target node t, such that $x = (G, s, t)$, and each response $y \in \mathcal{Y}$ ideally encodes an $s \to t$ path via a sequence of vertices $(s, v_1, \ldots, v_n, t)$. In our experiments we fix a single prompt distribution $\mu \in \Delta(\mathcal{X})$ for both pre-training and SFT stages, however we use different data-collection policies $\pi_{\mathrm{pre}}, \pi_{\mathrm{SFT}} : Y \to \Delta(\mathcal{X})$, both of which map prompts to (distributions over) $s \to t$ paths.

All graphs G are directed layered graphs with 10 total layers and with edges only between consecutive layers. The source vertex s is the only vertex in layer $L = 1$ and the target vertex t is the only vertex in layer 10. The subsequent 8 layers have 4 vertices each. For each of layers $L \in \{1, \ldots, 9\}$, a subset of the vertices in that layer are called *passable*. The edge structure is such that every passable node in layer ℓ has directed edges to all nodes in layer $\ell + 1$. Nodes that are not passable have no out-going edges. Thus a valid $s \to t$ path consists of 10 total nodes $(s, v_1, \ldots, v_8, t)$ where v_ℓ is in layer ℓ and each v_ℓ must be passable.

Graphs are parameterized by a number $k \in \{0, \ldots, 8\}$ denoting the number of layers with *two* passable nodes; the remaining layers have exactly *one* passable node. Given k we choose the layers with two passable nodes at random. Choosing the passable nodes is more intricate. We identify every vertex with a number $\in \{0, \ldots, 99\}$, such that in each of the intermediate layers, at least one vertex is *odd* and at least one vertex is *even*. When there is just one passable vertex in a layer, it is chosen uniformly at random. When there are two passable vertices, they are chosen at random such that one is odd and one is even. Thus, there are 2^k valid $s \to t$ paths in each graph. We use $k = 3$ for all experiments.

As described above, pre-training and SFT share the same prompt distribution. These are graphs of the above structure with vertex names assigned randomly (subject to the aforementioned even/odd restrictions in each layer), with passable nodes assigned randomly, and with layers with two passable nodes selected randomly. For pre-training, the data-collection policy π_{pre} selects an $s \to t$ path uniformly at random from all 2^k valid paths.

For SFT, the data-collection policy is more complex. First the policy computes a certain "cryptographic" function of the input graph to determine if the graph belongs to SHARD0 or SHARD1 (specifically, the cryptographic function is the and of the parity of the vertices layers 1-5 and the parity of the vertices in layers 6-10). The shard determines a certain rule for how the policy chooses vertices in the layers where there are two passable nodes. Specifically in SHARD0 the policy alternates between agreeing with the parity of the target vertex t and disagreeing with the parity of target node t, such that it agrees with the target vertex parity in the first layer with two passable nodes, disagrees in the second, and so on. In SHARD1 the policy does exactly the opposite, it disagrees with the target vertex parity in the first layer with two passable nodes, agrees in the second, and so on.

Intuitively, learning the pre-training policy's response distribution is relatively easy as it only requires local rules such as identifying all passable nodes in the subsequent layer and choosing one at random. Pre-training on this distribution teaches the model to understand the input format and the high-level task. On the other hand, learning the SFT policy's response distribution is very challenging, as it requires identifying global structure both to determine the shard and to determine which passable vertex to select in each layer.

To measure the performance of the trained model, we observe that the SFT policy is deterministic (for any input graph x, there is a single path in the support of the SFT policy's distribution $\pi_{\text{SFT}}(\cdot \mid x)$) and set the reward to be 1 if and only if the path chosen by the trained model matches the path selected by the SFT policy. Since learning the SFT policy's response distribution is challenging, it is corresponding quite challenging to achieve high reward in this task.

Inputs and outputs are represented as follows. First, all vertices, including the source and target, are assigned a number in $\{0, \ldots, 99\}$. Then, the input prompt is represented as an edge list followed by the source and target nodes, formatted as:

$$x: \texttt{ u\_1 v\_1 | u\_2 v\_2 | . . . | u\_k v\_k / s t =}$$

where u_i, v_i are the numerical values assigned to the vertices in the ith edge. We use the delimiter characters `|`, `/`, and `=` to separate edges from each other, the edge list from the source and target vertices, and the input from the response, respectively. Responses y are formatted as:

$$y: \texttt{ v\_1 v\_2 v\_2 v\_3 . . . v\_9 v\_10.}$$

C.2 Training Details

We use the same numerical tokenizer and transformer architecture as Chen et al. (2026a). For tokenization, each node is v is tokenized as its numerical value and the special characters are tokenized as $100, 101, 102$. The architecture is a GPT2-style transformer model with 4 heads, 6 transformer blocks and a 384 dimensional embeddings. We used absolute positional encodings. We always use the Adam optimizer with a fixed learning(1×10^{-4} during pre-training and 5×10^{-6} during SFT) and a batch size of 128. Pre-training operates in an offline training setting with a fixed pool of 256,000 graphs while SFT operates in an online training setting with new graphs generated on-the-fly for each batch. We pre-train for 200k training steps and SFT for 50k training steps with evaluation every 200 steps.

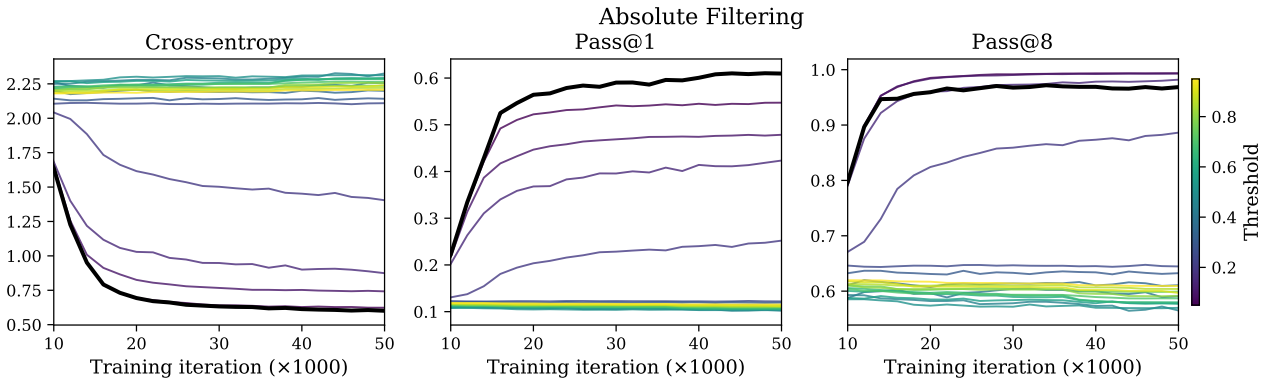

Figure 5: Detailed results of hyperparameter sweep for TailSFT with absolute clipping in the synthetic graph navigation experiment.

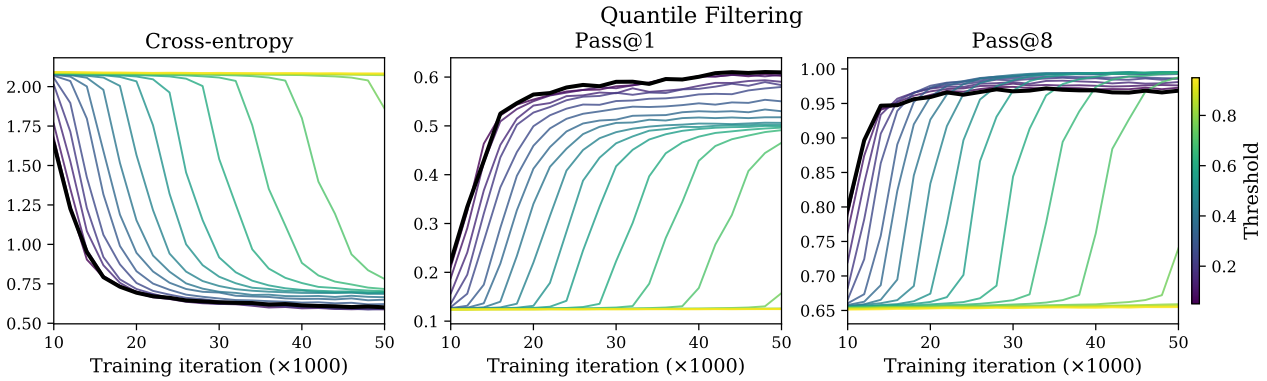

Figure 6: Detailed results of hyperparameter sweep for TailSFT with quantile clipping in the synthetic graph navigation experiment.

C.3 Additional Experimental Results

In Figure 3 we show the cross entropy, pass@1, and pass@8 for each of four methods: vanilla SFT, TailSFT with absolute clipping, TailSFT with quantile clipping, and TailSFT with offset quantile clipping using the pre-trained model as the reference. For the TailSFT variants we sweep over clipping hyperparameter and select the best hyperparameter based on pass@8 performance at the end of training. The specific hyperparameter selected are 0.05 for absolute, 0.4 for quantile, and 0.35 for offset quantile.

In Figs. 5 to 7, we show detailed results of the hyperparameter sweep for the three TailSFT variants. In all cases, we sweep the clipping hyperparameter in the range $0.05 \times \{1 \dots, 19\}$. For all variants clipping at value 0 corresponds to vanilla SFT as no sequences are dropped. We highlight two observations. First, absolute clipping is much more sensitive to the clipping hyperparameter than quantile or offset quantile variants. This is rather intuitive, as setting an absolute threshold for the cross-entropy loss requires understanding how the distribution of per-example cross-entropy losses evolves during training. On the other hand, setting the threshold as a per-batch quantile (either on the loss or on the offset loss) is much easier and these methods are much more robust. Indeed, the second observation is that for quantile-variants of TailSFT, setting the threshold too small recovers baseline SFT performance and setting the threshold too large does not impact pass@8 performance, but rather slows training convergence. This is also intuitive, as a large clipping parameter still focuses training on the harder examples but sacrifices computational efficiency as fewer per-sample gradients are utilized.

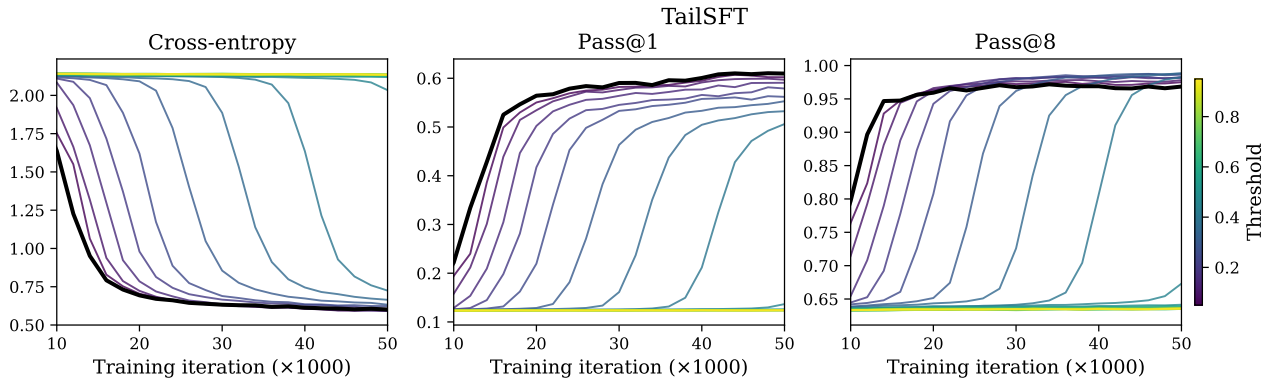

Figure 7: Detailed results of hyperparameter sweep for TailSFT with offset quantile clipping in the synthetic graph navigation experiment.

D Details for Language Modeling Experiments

D.1 SFT

The language modeling experiments use the same TailSFT algorithm (Algorithm 1) for both math and code but differ in data, prompt format, hyperparameter grid, and evaluation. We describe the shared procedure here; the code (Appendix D.1.1) and math (Appendix D.1.2) subsections give the domain-specific details.

Supervised fine-tuning. Every run starts from the OLMo-3 7B base model and minimizes the standard supervised cross-entropy loss on chat-formatted instruction/response pairs. We apply the OLMo-3 chat template and mask every prompt token, so only the final assistant turn and its terminating end-of-turn token count toward the loss. All runs use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and $\epsilon = 10^{-8}$, gradient-norm clipping at 1.0, 8-way data parallelism, and a learning rate held constant after a linear warmup over the first 3% of steps. Sequence length, weight decay, microbatch size, effective batch, epoch count, and the learning-rate grid are set per domain and given in each subsection. The per-step loss is length-normalized as in OLMo-3: we sum the target-token cross-entropy over the batch and divide by the number of unmasked target tokens, so every target token is weighted equally rather than every sequence.

Example filtering (TailSFT). TailSFT drops a fraction of the already-fit examples from the loss at each step. Before training we score every example x once with the base model, using the same tokenization and assistant-only mask as SFT, to get its initialization loss $\ell_0(x)$, the mean cross-entropy over the target tokens. During training we compare the current per-example loss $\ell_t(x)$ against this baseline through the signed margin

$$m_t(x) = \ell_t(x) - \ell_0(x).$$

A very negative margin means the loss on x has already dropped well below its starting value. We filter out the examples with the smallest (most negative) margins and train on the rest. Both ℓ_0 and ℓ_t are length-normalized per example—the mean cross-entropy over that example's target tokens—so the filtering decision does not favor longer or shorter responses.

The filtering decision runs on the *selection batch*, the $W \times b$ examples in a single forward pass across the $W = 8$ data-parallel ranks with per-rank microbatch b. The per-example losses are all-gathered so every rank computes the same mask. We do not filter over the full gradient-accumulated optimizer batch or over one rank's local microbatch; each accumulation step is filtered on its own selection batch. Within a selection batch we rank the examples by m_t and zero out the $\lfloor Wb\,f \rceil$ with the smallest margins, then train the survivors with the usual token-averaged cross-entropy. The code runs use $b = 1$ (an 8-example selection batch) and the math runs use $b = 2$ (a 16-example selection batch). Reproducing the method needs only the base-model losses ℓ_0, the filter fraction f, and its schedule; everything else is ordinary distributed SFT.

Filtering notation. A run is specified by its filter fraction $f \in [0, 1]$, the fraction of each selection batch that is dropped ($\lfloor Wb\,f \rceil$ examples), and a schedule for how f moves over training. A *static* schedule holds f

Table 3: Per-benchmark pass@1 for the OLMo-3 7B base model (no SFT), Standard SFT (f=0), and TailSFT, at the settings used in the main results table. $\Delta_{\text{Std}-\text{Base}}$ is Standard SFT minus base; $\Delta_{\text{Tail}-\text{Base}}$ is TailSFT minus base. Emdashes indicate missing evaluations.

SFT data	Benchmark	Base	Standard SFT	TailSFT	$\Delta_{\text{Std}-\text{Base}}$	$\Delta_{\text{Tail}-\text{Base}}$
OMI	AIME	5.16	$3.65_{\pm 0.37}$	$4.03_{\pm 0.02}$	−1.51	−1.13
	MATH-500 L5	—	$24.80_{\pm 0.40}$	$25.16_{\pm 0.40}$	—	—
	OMEGA-500	$3.54_{\pm 0.30}$	$6.58_{\pm 0.50}$	$6.51_{\pm 0.26}$	+3.04	+2.97
BigCode	MBPP+	$36.82_{\pm 0.56}$	$53.04_{\pm 0.93}$	$51.36_{\pm 0.14}$	+16.22	+14.54
	HumanEval+	$32.94_{\pm 1.11}$	$45.06_{\pm 0.57}$	$46.33_{\pm 1.07}$	+12.12	+13.39
	CruxEval-I	$25.67_{\pm 0.62}$	$29.79_{\pm 0.48}$	$30.43_{\pm 0.15}$	+4.12	+4.76
	CruxEval-O	$4.22_{\pm 0.74}$	$4.16_{\pm 0.44}$	$13.18_{\pm 0.30}$	−0.06	+8.96
	LiveCodeBench	$5.50_{\pm 0.42}$	$10.74_{\pm 0.46}$	$11.49_{\pm 0.38}$	+5.24	+5.99
Magicoder	MBPP+	$36.82_{\pm 0.56}$	$55.35_{\pm 0.47}$	$54.60_{\pm 0.55}$	+18.53	+17.78
	HumanEval+	$32.94_{\pm 1.11}$	$48.49_{\pm 0.58}$	$47.69_{\pm 0.19}$	+15.55	+14.75
	CruxEval-I	$25.67_{\pm 0.62}$	$28.50_{\pm 0.13}$	$26.48_{\pm 0.45}$	+2.83	+0.81
	CruxEval-O	$4.22_{\pm 0.74}$	$12.86_{\pm 0.19}$	$16.16_{\pm 0.53}$	+8.64	+11.94
	LiveCodeBench	$5.50_{\pm 0.42}$	$14.84_{\pm 0.11}$	$14.32_{\pm 0.38}$	+9.34	+8.82
OCI	MBPP+	$36.82_{\pm 0.56}$	$58.26_{\pm 0.88}$	$55.81_{\pm 0.06}$	+21.44	+18.99
	HumanEval+	$32.94_{\pm 1.11}$	$54.76_{\pm 1.16}$	$51.94_{\pm 1.83}$	+21.82	+19.00
	CruxEval-I	$25.67_{\pm 0.62}$	$29.46_{\pm 0.46}$	$30.17_{\pm 0.64}$	+3.79	+4.50
	CruxEval-O	$4.22_{\pm 0.74}$	$9.30_{\pm 0.49}$	$10.84_{\pm 0.37}$	+5.08	+6.62
	LiveCodeBench	$5.50_{\pm 0.42}$	$12.24_{\pm 0.36}$	$11.49_{\pm 0.84}$	+6.74	+5.99

fixed. A *ramp* schedule written $0 \to f$ raises the fraction linearly from 0 at the first step to f at the last, using every example early and filtering hardest late. Standard SFT is $f = 0$. We describe every run by its filter fraction f and schedule in the below results.

Filtering rarely hurts. The consistent finding across both domains is that filtering preserves pass@16 coverage. At matched settings it almost always matches or improves the no-filter pass@16, the gains are sometimes large, and the only cost is a small pass@1 decrease from deprioritizing examples the model already fits. Each subsection reports the full per-benchmark grids and tallies how rarely filtering regresses coverage. Tables 3 and 4 collect the base model, Standard SFT, and TailSFT side by side at the main-table settings for pass@1 and pass@16. In several cases where standard SFT marginally lowers pass@16 below the base model, TailSFT rescues the coverage, recovering it back above the base level. If we instead select the best pass@16 checkpoint for each of Standard SFT and TailSFT, TailSFT still matches or improves coverage on most benchmarks.

The gains are broad, not concentrated. Table 5 decomposes each pass@16 gain into per-problem paired changes, showing that the improvements are spread across many problems rather than driven by a handful.

D.1.1 Code

Model and optimization. Code SFT trains at sequence length 2048, which effectively never truncates training targets. Runs use no weight decay and a per-rank microbatch of 1 with 4 gradient-accumulation steps, for an effective batch of 32 and an 8-example selection batch. The learning rate is searched over $\{1, 2, 3\} \times 10^{-5}$. Each dataset uses a fixed epoch count.

Magicoder. We keep the Python subset of `ise-uiuc/Magicoder-OSS-Instruct-75K` (38,284 of 75,197 rows). We drop non-Python rows, rows without extractable code, rows matching an MBPP+ or HumanEval+ signature, and rows containing unit tests, and discard solutions that fail to parse as an abstract syntax tree. The assistant target starts with the prefix `Here is the completed function:\n\n“‘python\n`, followed by the extracted code and a closing fence, matching the MBPP+ and HumanEval+ answer format. One further row is removed because it does not fit the 2048-token training tokenization, leaving 33,817 train rows.

Table 4: Per-benchmark pass@16 (coverage) for the OLMo-3 7B base model (no SFT), Standard SFT (f=0), and TailSFT, at the settings used in the main results table. Columns as in Table 3.

SFT data	Benchmark	Base	Standard SFT	TailSFT	$\Delta_{\text{Std}-\text{Base}}$	$\Delta_{\text{Tail}-\text{Base}}$
OMI	AIME	20.91	$15.24_{\pm 2.02}$	$18.31_{\pm 0.57}$	−5.67	−2.60
	MATH-500 L5	—	$66.42_{\pm 0.00}$	$69.15_{\pm 0.86}$	—	—
	OMEGA-500	$26.13_{\pm 1.42}$	$32.80_{\pm 2.25}$	$32.60_{\pm 1.56}$	+6.67	+6.47
BigCode	MBPP+	$75.93_{\pm 1.06}$	$74.69_{\pm 1.10}$	$78.84_{\pm 1.06}$	−1.24	+2.91
	HumanEval+	$79.27_{\pm 0.61}$	$76.63_{\pm 0.70}$	$80.08_{\pm 0.93}$	−2.64	+0.81
	CruxEval-I	$70.50_{\pm 1.54}$	$61.71_{\pm 2.27}$	$65.79_{\pm 0.26}$	−8.79	−4.71
	CruxEval-O	$32.08_{\pm 1.98}$	$24.21_{\pm 2.38}$	$41.00_{\pm 1.35}$	−7.87	+8.92
	LiveCodeBench	$32.30_{\pm 0.93}$	$30.39_{\pm 0.86}$	$33.12_{\pm 0.34}$	−1.91	+0.82
Magicoder	MBPP+	$75.93_{\pm 1.06}$	$75.31_{\pm 0.93}$	$78.66_{\pm 1.25}$	−0.62	+2.73
	HumanEval+	$79.27_{\pm 0.61}$	$77.85_{\pm 0.70}$	$78.66_{\pm 1.83}$	−1.42	−0.61
	CruxEval-I	$70.50_{\pm 1.54}$	$59.75_{\pm 0.62}$	$68.08_{\pm 0.94}$	−10.75	−2.42
	CruxEval-O	$32.08_{\pm 1.98}$	$38.08_{\pm 0.94}$	$47.92_{\pm 1.66}$	+6.00	+15.84
	LiveCodeBench	$32.30_{\pm 0.93}$	$31.81_{\pm 0.34}$	$33.22_{\pm 0.34}$	−0.49	+0.92
OCI	MBPP+	$75.93_{\pm 1.06}$	$81.22_{\pm 0.75}$	$82.36_{\pm 0.81}$	+5.29	+6.43
	HumanEval+	$79.27_{\pm 0.61}$	$85.98_{\pm 1.61}$	$83.23_{\pm 2.16}$	+6.71	+3.96
	CruxEval-I	$70.50_{\pm 1.54}$	$68.08_{\pm 1.77}$	$71.58_{\pm 1.91}$	−2.42	+1.08
	CruxEval-O	$32.08_{\pm 1.98}$	$40.12_{\pm 0.00}$	$44.46_{\pm 0.47}$	+8.04	+12.38
	LiveCodeBench	$32.30_{\pm 0.93}$	$34.59_{\pm 0.90}$	$33.50_{\pm 0.16}$	+2.29	+1.20

BigCode. For `bigcode/self-oss-instruct-sc2-exec-filter-50k` (50,661 raw rows) the user message is the instruction and the target uses the same evalplus-aligned prefix and fenced Python format as Magicoder. We remove exact and near MBPP+ and HumanEval+ contamination, benchmark-signature contamination, rows containing tests, and non-Python rows, leaving 49,260 train rows.

OCI. For `nvidia/OpenCodeInstruct` we load five evenly spaced shards and keep rows with `average_test_score`= 1.0. We remove MBPP+ contamination by exact prompt match, by test-assertion substring match against inputs, outputs, and unit tests, and by MBPP+ function-name definitions appearing in outputs, then shuffle with seed 42 and hold out a 0.5% validation split. OCI keeps its native format, with the user message the OCI `input` and the target the OCI `output`, both verbatim.

Hyperparameter configurations. We vary the learning rate and filtering configuration as summarized in Table 7. The configurations used in Table 1 and the additional evaluated configurations are reported across several tables. Tables 8 and 9 give MBPP+ pass@1 and pass@16 for every completed BigCode and Magicoder configuration, with the change against the no-filter run at the same learning rate.

For OCI, Table 10 reports the no-filter Standard configuration and the TailSFT ramp configuration used in Table 1. MBPP+ coverage improves under filtering (+1.15 pp), while HumanEval+ loses coverage (−2.74 pp). Accordingly, OCI HumanEval+ has $\rho_{16} = 0.23$, outside the $\rho_{16} > 1$ regime identified by the coverage-ratio diagnostic.

Coverage holds across benchmarks and can repair coverage collapse. Tables 11–14 report every filtered configuration we evaluated on HumanEval+, CruxEval-I, CruxEval-O, and LiveCodeBench, against the no-filter run for the same dataset. The largest coverage gains land where standard SFT had driven coverage *below* the base model: on CruxEval-I standard SFT falls 8–11 pp under the base (70.5 → 61.7 for BigCode, 59.8 for Magicoder), so much of TailSFT's advantage is repairing a coverage collapse that ordinary SFT introduced.

Evaluation protocol. We evaluate with the OLMES (Gu et al., 2025) EvalPlus configuration used across the OLMo-3 code suite, so the appendix numbers are directly comparable to that standard. Every code result is sampled at temperature 1.0, top-p = 1.0, with 16 generations per problem for each of three independent evaluation seed sets; throughout the appendix, reported means and standard deviations are computed over these three seed sets, which we refer to simply as three seeds. MBPP+ uses the 378-problem EvalPlus test

SFT data	Benchmark	#gained	#lost	#unchanged	median Δ (pp)	mean Δ (pp)	95% CI
OMI	AIME	31	14	75	+0.00	+3.07	[+0.77, +5.59]
OMI	MATH-500 L5	25	15	94	+0.00	+2.74	[-1.24, +6.47]
OMI	OMEGA-500	75	77	348	+0.00	-0.20	[-2.33, +1.80]
BigCode	MBPP+	39	19	320	+0.00	+4.14	[+2.12, +6.35]
BigCode	HumanEval+	21	11	132	+0.00	+3.46	[+0.00, +6.91]
BigCode	CruxEval-I	157	89	554	+0.00	+4.08	[+2.12, +6.12]
BigCode	CruxEval-O	274	44	482	+0.00	+16.79	[+14.54, +19.04]
BigCode	LiveCodeBench	72	41	499	+0.00	+2.72	[+1.09, +4.41]
Magicoder	MBPP+	42	19	317	+0.00	+3.35	[+1.06, +5.73]
Magicoder	HumanEval+	13	10	141	+0.00	+0.81	[-2.03, +3.86]
Magicoder	CruxEval-I	196	82	522	+0.00	+8.33	[+6.17, +10.46]
Magicoder	CruxEval-O	183	48	569	+0.00	+9.83	[+7.92, +11.83]
Magicoder	LiveCodeBench	56	41	515	+0.00	+1.42	[-0.22, +3.05]
OCI	MBPP+	22	15	341	+0.00	+1.15	[-0.35, +2.60]
OCI	HumanEval+	8	17	139	+0.00	-2.74	[-5.18, -0.51]
OCI	CruxEval-I	134	69	597	+0.00	+3.50	[+2.04, +4.92]
OCI	CruxEval-O	132	57	611	+0.00	+4.33	[+2.83, +5.79]
OCI	LiveCodeBench	45	65	502	+0.00	-1.09	[-2.61, +0.44]

Table 5: Per-problem paired changes in pass@16 for TAILSFT versus Standard SFT at the main-table settings. A problem counts as gained or lost only when its paired pass@16 strictly changes; unchanged includes ceiling and floor ties. The final column is a paired bootstrap 95% interval for the mean per-problem change.

SFT data	Source	Train / val
Magicoder	`ise-uiuc/Magicoder-OSS-Instruct-75K`	33,817 / 342
BigCode	`bigcode/self-oss-instruct-sc2-exec-filter-50k`	49,260 / 498
OCI	`nvidia/OpenCodeInstruct`	72,635 / 365

Table 6: Code SFT datasets. Row counts are after decontamination, format conversion, and initialization-loss annotation.

split with the assistant prefix `Here is the completed function:\n\n“‘python\n`, maximum generation length 2048, context length 4096, and stop strings “‘, newline-triple-quote, newline-`assert`, and newline-comment. HumanEval+ uses the 164-problem EvalPlus split with the same prefix, generation length 1024, and context 4096. CruxEval-I and CruxEval-O use 800 instances each with OLMo-3 multiturn few-shot prompts (two shots for input prediction, one for output), generation length 512, and context 4096. LiveCodeBench uses 612 code-generation problems with an expert-Python system prompt, generation length 8192, and context 32768. We report pass@k with the standard unbiased estimator implemented in OLMES, pass@$k = 1 - \binom{n-c}{k} / \binom{n}{k}$ for a document with c correct completions out of n samples, averaged over documents rather than as a best-of-k maximum. With n=16 samples this makes pass@1 the mean per-document pass rate and pass@16 the fraction of documents with at least one correct completion. Each generation length sits above the lengths these tasks actually decode to, so completions are not cut off.

The coverage advantage tends to widen with k. Table 16 reports the TAILSFT minus Standard SFT pass@k gap at $k \in \{1, 2, 4, 8, 16\}$ for every benchmark. On most cells the gap starts near zero (or slightly negative) at k=1 and tends to increase with k—most strikingly CruxEval-O, where BigCode grows from +9.02 to +16.79—so the advantage is a coverage effect rather than a pass@1 shift. The two OCI transfer benchmarks (HumanEval+ and LiveCodeBench) are the main exception, where TAILSFT trails at every k.

D.1.2 Math

Model and optimization. Math SFT trains at sequence length 4096 with weight decay 0.1, a per-rank microbatch of 2 with 4 gradient-accumulation steps, for an effective batch of 64 and a 16-example selection batch. The learning rate is fixed at 3×10^{-5}. We train for 2 epochs and evaluate the final checkpoint. Two passes fit the $\sim$350k-example subset without the overfitting seen at higher counts, and the count is held identical across every arm so differences come only from filtering. Each supervised example is a two-turn

Hyperparameter	Values
Base model	OLMo-3 7B base
Learning rate	$\{1, 2, 3\} \times 10^{-5}$
Filter schedule	none; static; ramp $0 \to \cdot$
Filter fraction f	0 (none); static $\{0.125, 0.25, 0.5\}$; ramp $0 \to 0.5$
Epochs	fixed per dataset (4 / 5 / 3 for BigCode / Magicoder / OCI)
Sequence length	2048
Effective batch	32 ($8 \times 1 \times 4$)
Warmup / schedule	3% linear warmup, then constant
Optimizer	AdamW $(0.9, 0.95)$, wd 0, clip 1.0

Table 7: Code SFT hyperparameter grid over learning rates and filtering configurations.

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	36.82 ± 0.56	—	75.93 ± 1.06	—
no filter	1×10^{-5}	47.45 ± 0.82	—	79.45 ± 0.81	—
static f=0.125	1×10^{-5}	47.00 ± 0.60	−0.45	79.81 ± 0.81	+0.36
ramp 0→0.5	1×10^{-5}	45.62 ± 0.57	−1.83	79.72 ± 0.85	+0.27
no filter	2×10^{-5}	52.58 ± 0.11	—	76.72 ± 1.06	—
static f=0.125	2×10^{-5}	52.07 ± 0.70	−0.51	77.51 ± 0.46	+0.79
static f=0.25	2×10^{-5}	51.36 ± 0.14	−1.22	78.84 ± 1.06	+2.12
static f=0.5	2×10^{-5}	47.22 ± 0.54	−5.36	77.95 ± 0.76	+1.23
ramp 0→0.5	2×10^{-5}	51.65 ± 0.18	−0.93	78.66 ± 0.40	+1.94
no filter	3×10^{-5}	53.04 ± 0.93	—	74.69 ± 1.10	—
static f=0.125	3×10^{-5}	51.90 ± 0.59	−1.14	75.40 ± 1.06	+0.71
ramp 0→0.5	3×10^{-5}	50.67 ± 0.26	−2.37	75.93 ± 0.46	+1.24

Table 8: BigCode MBPP+ (n=378) SFT filtering grid, 4 epochs. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run at the same learning rate. The *base (no SFT)* row is the untuned OLMo-3 7B base under the same MBPP+ protocol (format-robust EvalPlus stops).

chat. The user message is the problem followed by the instruction "`Present the answer in LaTex format: \boxed{Your answer}`", and the assistant target is the provided chain-of-thought solution.

Data. The OpenMathInstruct-2 (OMI) subset is built from the OMI `train_1M` shards by keeping `problem_source` in {`math`, `augmented_math`}, exact-match decontaminating against MATH-500 problem strings, dropping rows whose problem plus solution exceeds 2048 tokens under the OLMo-3 tokenizer, and sampling a 350k-example stratified subset with a 95/5 train/validation split. No MATH-500 rows were found during decontamination. Since rows over 2048 tokens were removed during data construction, the 4096-token training sequence length does not truncate supervised targets. The final SFT parquets add `init_ce`, the base-model per-example cross-entropy used by the filtering rule.

Hyperparameter search and selection. For math we fixed the learning rate at 3×10^{-5} and swept the filtering configuration over no filtering, static $f \in \{0.0625, 0.125, 0.1875, 0.25, 0.5\}$, and the linear ramps $0 \to f$ for the same endpoints. The optimizer, sequence length, batch size, and 2-epoch schedule are held fixed as shown in Table 18. Table 1 then reports the no-filter Standard checkpoint against the selected TailSFT checkpoint. We use the ramp $0 \to 0.5$ arm.

Per-benchmark grids. Tables 19–22 report every completed math SFT filtering evaluation in-domain math benchmarks. The broadest grid is MATH-500. Its 500-problem aggregate (Table 19) sits near ceiling, so Tables 20 and 21 break the same sweep out by the five MATH difficulty levels. No-filter pass@16 is 100% at Level 1 and stays above 96% through Level 3, so the easy levels dominate the aggregate and hide where filtering acts. The unreached coverage, and the gains from filtering, concentrate at the hardest levels: at Level 5 filtering matches or improves no-filter pass@16 in 8 of 10 arms and strictly improves it in 7, while the

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	36.82 ± 0.56	—	75.93 ± 1.06	—
no filter	1×10^{-5}	53.14 ± 0.52	—	80.69 ± 0.26	—
static f=0.125	1×10^{-5}	52.19 ± 0.25	−0.95	79.98 ± 0.61	−0.71
static f=0.25	1×10^{-5}	51.51 ± 0.04	−1.63	80.42 ± 1.40	−0.27
static f=0.5	1×10^{-5}	50.00 ± 0.59	−3.14	79.45 ± 1.55	−1.24
no filter	2×10^{-5}	55.01 ± 0.36	—	75.13 ± 1.15	—
static f=0.125	2×10^{-5}	54.83 ± 0.52	−0.18	77.51 ± 0.53	+2.38
static f=0.25	2×10^{-5}	54.60 ± 0.55	−0.41	78.66 ± 1.25	+3.53
static f=0.5	2×10^{-5}	51.92 ± 0.41	−3.09	79.72 ± 0.31	+4.59
ramp 0→0.5	2×10^{-5}	54.73 ± 0.73	−0.28	78.31 ± 1.47	+3.18
no filter	3×10^{-5}	55.35 ± 0.47	—	75.31 ± 0.93	—
static f=0.125	3×10^{-5}	54.75 ± 0.58	−0.60	76.54 ± 0.15	+1.23
static f=0.25	3×10^{-5}	52.94 ± 0.67	−2.41	75.93 ± 1.06	+0.62
static f=0.5	3×10^{-5}	51.14 ± 0.47	−4.21	77.95 ± 1.36	+2.64

Table 9: Magicoder MBPP+ (n=378) SFT filtering grid, 5 epochs. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run at the same learning rate. The *base (no SFT)* row is the same untuned base as in Table 8.

Benchmark	Filter	pass@1	Δ	pass@16	Δ
MBPP+ (n=378)	base (no SFT)	36.82 ± 0.56	—	75.93 ± 1.06	—
MBPP+ (n=378)	no filter	58.26 ± 0.88	—	81.22 ± 0.75	—
MBPP+ (n=378)	ramp 0→0.5	55.81 ± 0.06	−2.44	82.36 ± 0.81	+1.15
HumanEval+ (n=164)	base (no SFT)	32.94 ± 1.11	—	79.27 ± 0.61	—
HumanEval+ (n=164)	no filter	54.76 ± 1.16	—	85.98 ± 1.61	—
HumanEval+ (n=164)	ramp 0→0.5	51.94 ± 1.83	−2.82	83.23 ± 2.16	−2.74

Table 10: OCI MBPP+ and HumanEval+ SFT grid, 3 epochs, learning rate 2×10^{-5}; Standard is no filtering and TailSFT is the ramp 0→0.5. Values are percent, mean ± sample standard deviation; Δ is the change against no filtering. The *base (no SFT)* rows are the untuned base under the same per-benchmark protocol.

near-saturated levels move little in either direction. No MATH-500 pass@16 cell drops by more than 1.8 pp, inside the seed noise. The completed AIME rows all improve pass@16, while the frozen OMEGA-500 arm is essentially tied with no filtering.

Reading the Level-4 numbers. Level 4 is the one subset where filtering neither clearly helps nor hurts, and this is an estimator artifact rather than a real regression. pass@16 is scored as a binary hit per seed and averaged over three seeds, so every per-problem value is quantized to $\{0, \frac{1}{3}, \frac{2}{3}, 1\}$ and a problem's filtered-minus-no-filter difference can only be a multiple of 33 pp. The Level-4 mean is then set by a few knife's-edge problems that the base model already solves on two of three seeds, where a single seed flip moves the subset mean by a point or two. Two such problems account for the whole gap: under the arms that most help Level 5, the Level-4 pass@16 change moves from +0.0 pp on the full subset to about +0.8 pp once those two problems are excluded, in the same direction as Level 5. Level 5 shows a larger and steadier gain because it contains many more genuinely contested problems, so this per-problem quantization averages out.

Evaluation protocol. We evaluate with the standard OLMES (Gu et al., 2025) math configuration used across the OLMo-3 math suite. AIME contains 30 problems from each year 2022–2025 ($n = 120$), chat prompting with the boxed-answer instruction, temperature 0.6, top-$p = 0.95$, maximum generation length and model context 8192, and 32 generations per problem for each of three seeds. MATH-500 uses the 500-problem test split; the main table reports the Level-5 subset ($n = 134$), since the full benchmark is near ceiling, with temperature 1.0, top-$p = 1.0$, maximum generation length 8192, model context 16384, and 16 generations per problem for each of three seeds. OMEGA-500 (the `saumyamalik/omega-500` 500-problem set) uses temperature

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	32.94 ± 1.11	—	79.27 ± 0.61	—
BigCode					
no filter	3×10^{-5}	45.06 ± 0.57	—	76.63 ± 0.70	—
static f=0.125	2×10^{-5}	48.08 ± 0.12	+3.02	79.47 ± 1.76	+2.85
static f=0.25	2×10^{-5}	46.33 ± 1.07	+1.27	80.08 ± 0.93	+3.46
static f=0.5	2×10^{-5}	44.08 ± 0.48	−0.98	80.69 ± 1.27	+4.07
Magicoder					
no filter	3×10^{-5}	48.49 ± 0.58	—	77.85 ± 0.70	—
static f=0.25	2×10^{-5}	47.69 ± 0.19	−0.80	78.66 ± 1.83	+0.81
static f=0.5	2×10^{-5}	46.60 ± 0.66	−1.89	80.49 ± 1.61	+2.64

Table 11: HumanEval+ (n=164) SFT filtering grid. Epochs are 4 (BigCode) and 5 (Magicoder); OCI HumanEval+ appears in Table 10. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run for the same dataset; the *base (no SFT)* row is the untuned OLMo-3 7B base under the same protocol.

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	25.67 ± 0.62	—	70.50 ± 1.54	—
BigCode					
no filter	3×10^{-5}	29.79 ± 0.48	—	61.71 ± 2.27	—
static f=0.125	2×10^{-5}	30.65 ± 0.26	+0.86	63.54 ± 0.76	+1.83
static f=0.25	2×10^{-5}	30.43 ± 0.15	+0.64	65.79 ± 0.26	+4.08
static f=0.5	2×10^{-5}	29.76 ± 0.35	−0.03	69.29 ± 1.92	+7.58
Magicoder					
no filter	3×10^{-5}	28.50 ± 0.13	—	59.75 ± 0.62	—
static f=0.25	2×10^{-5}	26.48 ± 0.45	−2.02	68.08 ± 0.94	+8.33
static f=0.5	2×10^{-5}	25.63 ± 0.56	−2.88	69.79 ± 0.38	+10.04
OCI					
no filter	2×10^{-5}	29.46 ± 0.46	—	68.08 ± 1.77	—
ramp 0→0.5	2×10^{-5}	30.17 ± 0.64	+0.71	71.58 ± 1.91	+3.50

Table 12: CruxEval-I (n=800) SFT filtering grid. Epochs are 4 / 5 / 3 for BigCode / Magicoder / OCI. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run for the same dataset; the *base (no SFT)* row is the untuned OLMo-3 7B base under the same protocol.

1.0, top-p = 1.0, maximum generation length 4096, model context 8192, and 16 generations per problem for each of three seeds. For AIME and MATH-500 we use the stored per-document pass@k metrics. For OMEGA-500, whose aggregate files do not store pass@k, we recompute pass@1 and pass@16 from the 16 completions using the extracted flexible boxed answer, falling back to the standard extracted answer, and match it against the label. In every case pass@k is the same unbiased estimator implemented in OLMES (pass@$k = 1 - \binom{n-c}{k} / \binom{n}{k}$ for c correct of n samples), so AIME's 32 samples are combined at k=16 rather than by best-of-k. The sweep of Table 19 uses MATH-500 full ($n = 500$) with 4096-token generations.

The coverage advantage tends to widen with k. Table 24 reports the TailSFT minus Standard SFT pass@k gap at $k \in \{1, 2, 4, 8, 16\}$. On AIME and MATH-500 the gap is small at k=1 and tends to grow with k (AIME +0.37 → +3.07; MATH-500 Level 5 +0.36 → +2.74), while OMEGA-500, where the two models are already matched, stays flat. The math gains therefore come from broader coverage that surfaces at larger k rather than from a shift in greedy accuracy.

D.2 GRPO

The in-domain math and code GRPO runs share one training procedure and differ only in data, reward, hyperparameter grid, and evaluation. We describe the shared procedure here; the math (Appendix D.2.1)

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	4.22 ± 0.74	—	32.08 ± 1.98	—
BigCode					
no filter	3×10^{-5}	4.16 ± 0.44	—	24.21 ± 2.38	—
static f=0.125	2×10^{-5}	12.02 ± 0.11	+7.86	39.75 ± 0.76	+15.54
static f=0.25	2×10^{-5}	13.18 ± 0.30	+9.02	41.00 ± 1.35	+16.79
static f=0.5	2×10^{-5}	7.35 ± 0.37	+3.20	36.54 ± 0.14	+12.33
Magicoder					
no filter	3×10^{-5}	12.86 ± 0.19	—	38.08 ± 0.94	—
static f=0.25	2×10^{-5}	16.16 ± 0.53	+3.30	47.92 ± 1.66	+9.83
static f=0.5	2×10^{-5}	10.12 ± 0.37	−2.74	41.67 ± 0.63	+3.58
OCI					
no filter	2×10^{-5}	9.30 ± 0.49	—	40.12 ± 0.00	—
ramp 0→0.5	2×10^{-5}	10.84 ± 0.37	+1.54	44.46 ± 0.47	+4.33

Table 13: CruxEval-O (n=800) SFT filtering grid. Epochs are 4 / 5 / 3 for BigCode / Magicoder / OCI. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run for the same dataset; the *base (no SFT)* row is the untuned OLMo-3 7B base under the same protocol.

Filter	LR	pass@1	Δ	pass@16	Δ
base (no SFT)	—	5.50 ± 0.42	—	32.30 ± 0.93	—
BigCode					
no filter	3×10^{-5}	10.74 ± 0.46	—	30.39 ± 0.86	—
static f=0.125	2×10^{-5}	11.77 ± 0.44	+1.03	31.54 ± 0.71	+1.14
static f=0.25	2×10^{-5}	11.49 ± 0.38	+0.75	33.12 ± 0.34	+2.72
static f=0.5	2×10^{-5}	11.45 ± 0.42	+0.71	33.39 ± 0.25	+3.00
Magicoder					
no filter	3×10^{-5}	14.84 ± 0.11	—	31.81 ± 0.34	—
static f=0.25	2×10^{-5}	14.32 ± 0.38	−0.51	33.22 ± 0.34	+1.42
static f=0.5	2×10^{-5}	14.81 ± 0.22	−0.02	34.48 ± 1.28	+2.67
OCI					
no filter	2×10^{-5}	12.24 ± 0.36	—	34.59 ± 0.90	—
ramp 0→0.5	2×10^{-5}	11.49 ± 0.84	−0.75	33.50 ± 0.16	−1.09

Table 14: LiveCodeBench (n=612) SFT filtering grid. Epochs are 4 / 5 / 3 for BigCode / Magicoder / OCI. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the no-filter run for the same dataset; the *base (no SFT)* row is the untuned OLMo-3 7B base under the same protocol.

and code (Appendix D.2.2) subsections give the domain-specific details. Every run is initialized from one of the matched SFT checkpoints evaluated in Table 1: the Standard-SFT run supplies the no-filter start and the paired TailSFT run supplies the filtered start, so the two policies differ only in how their initialization was trained. Table 2 reports a single tuned configuration per domain; the per-domain grids below give the tuned-configuration results in full.

Reinforcement learning with GRPO. Each run optimizes the initialized policy against a verifiable reward with GRPO (Shao et al., 2024). For every prompt we sample a group of n responses at temperature 1.0, score each response with the domain reward, and use the group-relative advantage—each response's reward centered and scaled within its group—as the policy-gradient signal. We use no KL regularization and no learned reward model. Optimization uses a batch of 128 prompts, a mini-batch of 128, and 8-way data parallelism, with the rollout group size n and the actor learning rate searched per domain and all other settings held fixed within a domain. Checkpoints are exported periodically during training and the reported checkpoint is chosen on a held-out validation split, as described per domain. Post-GRPO evaluation uses OLMES (Gu et al., 2025) and reuses the same per-benchmark coverage protocols as Appendix D.1, with pass@k estimated over three seeds; the exact per-benchmark sampling settings are given below.

SFT data	Benchmark	Standard p@1	TailSFT p@1	Standard p@16	TailSFT p@16
BigCode	MBPP+ ($n = 378$)	$53.04_{\pm 0.93}$	$51.36_{\pm 0.14}$	$74.69_{\pm 1.10}$	$78.84_{\pm 1.06}$
BigCode	HumanEval+ ($n = 164$)	$45.06_{\pm 0.57}$	$46.33_{\pm 1.07}$	$76.63_{\pm 0.70}$	$80.08_{\pm 0.93}$
BigCode	CruxEval-I ($n = 800$)	$29.79_{\pm 0.48}$	$30.43_{\pm 0.15}$	$61.71_{\pm 2.27}$	$65.79_{\pm 0.26}$
BigCode	CruxEval-O ($n = 800$)	$4.16_{\pm 0.44}$	$13.18_{\pm 0.30}$	$24.21_{\pm 2.38}$	$41.00_{\pm 1.35}$
BigCode	LiveCodeBench ($n = 612$)	$10.74_{\pm 0.46}$	$11.49_{\pm 0.38}$	$30.39_{\pm 0.86}$	$33.12_{\pm 0.34}$
Magicoder	MBPP+ ($n = 378$)	$55.35_{\pm 0.47}$	$54.60_{\pm 0.55}$	$75.31_{\pm 0.93}$	$78.66_{\pm 1.25}$
Magicoder	HumanEval+ ($n = 164$)	$48.49_{\pm 0.58}$	$47.69_{\pm 0.19}$	$77.85_{\pm 0.70}$	$78.66_{\pm 1.83}$
Magicoder	CruxEval-I ($n = 800$)	$28.50_{\pm 0.13}$	$26.48_{\pm 0.45}$	$59.75_{\pm 0.62}$	$68.08_{\pm 0.94}$
Magicoder	CruxEval-O ($n = 800$)	$12.86_{\pm 0.19}$	$16.16_{\pm 0.53}$	$38.08_{\pm 0.94}$	$47.92_{\pm 1.66}$
Magicoder	LiveCodeBench ($n = 612$)	$14.84_{\pm 0.11}$	$14.32_{\pm 0.38}$	$31.81_{\pm 0.34}$	$33.22_{\pm 0.34}$
OCI	CruxEval-I ($n = 800$)	$29.46_{\pm 0.46}$	$30.17_{\pm 0.64}$	$68.08_{\pm 1.77}$	$71.58_{\pm 1.91}$
OCI	CruxEval-O ($n = 800$)	$9.30_{\pm 0.49}$	$10.84_{\pm 0.37}$	$40.12_{\pm 0.00}$	$44.46_{\pm 0.47}$
OCI	LiveCodeBench ($n = 612$)	$12.24_{\pm 0.36}$	$11.49_{\pm 0.84}$	$34.59_{\pm 0.90}$	$33.50_{\pm 0.16}$

Table 15: Post-SFT code results, matching Table 1. Values are percentages, mean ± sample standard deviation over three seeds.

Table 16: SFT pass@k deltas for code benchmarks, reported as TailSFT minus Standard SFT in percentage points at the main-table settings. Deltas are computed with the unbiased OLMES (Gu et al., 2025) pass@k estimator.

SFT data	Benchmark	Δp@1	Δp@2	Δp@4	Δp@8	Δp@16
BigCode	MBPP+ (n=378)	−1.68	+0.64	+2.19	+3.19	+4.14
	HumanEval+ (n=164)	+1.27	+3.37	+4.34	+4.44	+3.46
	CruxEval-I (n=800)	+0.64	+1.61	+2.57	+3.25	+4.08
	CruxEval-O (n=800)	+9.02	+12.58	+15.45	+16.95	+16.79
	LiveCodeBench (n=612)	+0.75	+1.39	+1.86	+2.21	+2.72
Magicoder	MBPP+ (n=378)	−0.75	+1.73	+2.95	+3.25	+3.35
	HumanEval+ (n=164)	−0.80	+0.81	+1.55	+1.64	+0.81
	CruxEval-I (n=800)	−2.02	−0.10	+2.36	+5.28	+8.33
	CruxEval-O (n=800)	+3.30	+4.97	+6.70	+8.38	+9.83
	LiveCodeBench (n=612)	−0.51	+0.30	+0.80	+1.08	+1.42
OCI	MBPP+ (n=378)	−2.44	−1.21	−0.17	+0.65	+1.15
	HumanEval+ (n=164)	−2.82	−2.38	−1.89	−2.02	−2.74
	CruxEval-I (n=800)	+0.71	+1.54	+2.46	+3.19	+3.50
	CruxEval-O (n=800)	+1.54	+2.33	+3.14	+3.86	+4.33
	LiveCodeBench (n=612)	−0.75	−0.86	−0.91	−0.91	−1.09

D.2.1 Math

Initialization. The Standard start is the no-filter OMI SFT checkpoint and the TailSFT start is the matched ramp 0→0.5 checkpoint (Appendix D.1.2); these are the two OMI checkpoints reported in Table 1.

Data and reward. GRPO trains on the MATH training split (the MATH-lighteval release) with the MATH-500 test problems removed by exact-match decontamination, leaving 11,396 training and 600 validation prompts, so the evaluation set is never seen during RL. Each prompt uses the same boxed-answer instruction as math SFT, and the reward is 1 when a response's boxed final answer matches the reference under the verifiable math grader and 0 otherwise.

Optimization. We use a maximum prompt length of 512, a maximum response length of 4096, and train for 3 epochs. At rollout group size n=4 we search the actor learning rate; Table 26 reports the tuned configuration (actor learning rate 2×10^{-5}). Doubling the rollout group size to n=8 leaves the ordering unchanged—the TailSFT initialization retains a post-GRPO pass@1 advantage of about +2 points on MATH-500 Level 5—so the gain is not an artifact of the group size. Table 25 summarizes the configuration.

Checkpoint selection and evaluation. We select the mean-best checkpoint—the one with the highest mean validation reward over the sampled group—which averages over the small held-out validation split to

Dataset	Source/filter	Train rows	Validation rows
OMI math subset	OpenMathInstruct-2 `math`/`augmented_math`, MATH-500 decontaminated	332,500	17,500

Table 17: In-domain math SFT data. The annotated training split contains 326,725 `augmented_math` and 5,775 `math` rows; the validation split contains 17,192 and 308, respectively.

Hyperparameter	Values
Base model	OLMo-3 7B base
Learning rate	fixed at 3×10^{-5}
Filter schedule	none; static; ramp $0 \rightarrow \cdot$
Filter fraction f	0 (none); static $\{0.0625, 0.125, 0.1875, 0.25, 0.5\}$; ramp $0 \rightarrow \{0.0625, 0.125, 0.1875, 0.25, 0.5\}$
Epochs	2
Sequence length	4096
Effective batch	64 ($8 \times 2 \times 4$)
Warmup schedule	3% linear warmup, then constant
Optimizer	AdamW (0.9, 0.95), wd 0.1, clip 1.0

Table 18: Math SFT hyperparameter grid.

reduce selection noise. We evaluate on the MATH-500 Level-5 subset (n=134) and AIME 2022–2025 (n=120). Because these are long-form reasoning tasks we use a generation budget of 8192 tokens and report only these long-generation evaluations, since shorter budgets truncate reasoning traces and understate pass rates; at 8192 tokens truncation is not a concern, as no AIME generation and fewer than 4% of MATH-500 generations reach the limit. Matching the math SFT protocol (Appendix D.1.2), MATH-500 Level 5 is sampled at temperature 1.0, top-p=1.0 with 16 samples per problem, and AIME at temperature 0.6, top-p=0.95 with 32 samples per problem, each over three seeds. At the tuned configuration the TailSFT initialization improves both pass@1 and pass@16 on both benchmarks (Table 26).

D.2.2 Code

Initialization. For each dataset the Standard start is the no-filter SFT checkpoint and the TailSFT start is the matched filtered checkpoint from Appendix D.1.1 (static f=0.25 for BigCode and Magicoder, the ramp $0 \rightarrow 0.5$ for OCI); these are the checkpoints reported in Table 1.

Data and reward. GRPO trains on the MBPP+ training split with a held-out 100-problem validation subset. The reward is 1 when a sampled program passes the full MBPP+ unit-test suite for its problem and 0 otherwise.

Optimization. We use a maximum prompt and response length of 1024 each and train for 10 epochs. We search the rollout group size $n \in \{2, 4\}$ and the actor learning rate $\in \{5 \times 10^{-6}, 1 \times 10^{-5}, 2 \times 10^{-5}\}$; Table 28 reports the tuned configuration for each dataset. Table 27 summarizes the configuration.

Checkpoint selection and evaluation. We select the checkpoint with the highest validation pass rate over the small 100-problem held-out split, applying the same rule to both initializations. Evaluation uses EvalPlus MBPP+ (378-problem test split) at temperature 1.0, top-p=1.0, with 16 samples per problem over three seeds and a 2048-token generation budget; MBPP+ solutions are short relative to this budget, so generations terminate well within the limit and truncation does not affect the reported rates. At the tuned configurations the TailSFT initialization improves post-GRPO pass@1 while generally retaining its pass@16 coverage advantage (Table 28).

Training efficiency. Figure 8 tracks the mean training reward per GRPO step for the Standard and TailSFT initializations. The TailSFT reward rises at least as fast as Standard early in training, and in some settings the

Filter	pass@1	Δ	pass@16	Δ
no filter	53.55 ± 0.49	—	86.27 ± 0.61	—
static f=0.0625	53.66 ± 0.28	+0.11	87.00 ± 0.87	+0.73
static f=0.125	53.60 ± 0.39	+0.05	87.20 ± 0.40	+0.93
static f=0.1875	53.15 ± 0.33	−0.41	86.20 ± 0.69	−0.07
static f=0.25	53.51 ± 0.41	−0.04	86.33 ± 1.29	+0.07
static f=0.5	52.02 ± 0.13	−1.54	85.60 ± 0.72	−0.67
ramp 0→0.0625	53.80 ± 0.29	+0.25	85.87 ± 0.31	−0.40
ramp 0→0.125	53.37 ± 0.73	−0.19	86.47 ± 0.50	+0.20
ramp 0→0.1875	54.13 ± 0.50	+0.57	87.27 ± 0.95	+1.00
ramp 0→0.25	53.91 ± 0.19	+0.35	86.13 ± 0.46	−0.13
ramp 0→0.5	53.59 ± 0.41	+0.03	86.27 ± 1.01	+0.00

Table 19: MATH-500 full (n=500) SFT filtering grid, 4096-token generations. Values are percent, mean ± sample standard deviation over three generation seeds; Δ is the change against no filtering.

Filter	Level 1 (n=43)	Level 2 (n=90)	Level 3 (n=105)	Level 4 (n=128)	Level 5 (n=134)
no filter	100.00 ± 0.00	97.78 ± 1.11	96.51 ± 1.98	85.68 ± 1.63	66.67 ± 1.88
static f=0.0625	100.00 ± 0.00	98.15 ± 0.64	96.19 ± 0.95	85.68 ± 0.90	69.40 ± 3.25
static f=0.125	100.00 ± 0.00	97.41 ± 0.64	96.83 ± 1.10	85.68 ± 2.51	70.15 ± 1.29
static f=0.1875	100.00 ± 0.00	97.78 ± 0.00	95.56 ± 1.46	84.38 ± 0.78	68.41 ± 1.88
static f=0.25	100.00 ± 0.00	97.41 ± 0.64	96.51 ± 0.55	85.94 ± 3.40	66.92 ± 2.62
static f=0.5	100.00 ± 0.00	96.30 ± 1.70	95.56 ± 1.98	85.68 ± 3.25	65.92 ± 4.24
ramp 0→0.0625	99.22 ± 1.34	97.04 ± 0.64	97.14 ± 0.95	84.38 ± 0.78	66.67 ± 1.55
ramp 0→0.125	100.00 ± 0.00	97.04 ± 1.70	96.83 ± 1.46	83.85 ± 1.19	69.40 ± 2.69
ramp 0→0.1875	100.00 ± 0.00	97.41 ± 0.64	96.51 ± 0.55	85.68 ± 1.63	70.65 ± 4.97
ramp 0→0.25	100.00 ± 0.00	97.78 ± 1.11	97.78 ± 1.46	84.90 ± 0.45	65.92 ± 1.14
ramp 0→0.5	100.00 ± 0.00	96.67 ± 0.00	97.14 ± 0.00	85.42 ± 1.97	67.16 ± 1.97

Table 20: MATH-500 SFT **pass@16** by difficulty level, from the same 4096-token SFT filtering sweep as Table 19. Columns are the five MATH difficulty levels, with the number of problems in each header. Values are percent, mean ± sample standard deviation over three seeds. pass@16 is at or near ceiling through Level 3, so the aggregate is dominated by the easy levels; the gains from filtering concentrate at Levels 4–5. The frozen main table reports the selected ramp arm on Level 5, re-evaluated with 8192-token generations.

early reward increases up to roughly 2.5× faster.

D.3 Coverage ratio diagnostic: computation

This subsection records exactly how each point in Figure 4 was produced from the evaluation runs, filling in the aggregation and run-selection choices that Definition 4.1 leaves implicit. The generation settings (temperature, top-p, generation length, and sample count) for each benchmark are exactly those documented in Appendices D.1.1 and D.1.2; we do not re-specify them here.

Per-problem scoring and seed aggregation. For each problem i and model π we draw at least 16 samples per evaluation seed, compute the empirical pass@1 and pass@16 within each seed, and average those per-problem values across the available seeds to form $P_{i,1}(\pi)$ and $P_{i,16}(\pi)$; this is the $P_{i,K}(\pi)$ of Definition 4.1. Every run uses three seeds for base, standard, and TailSFT on every benchmark.

Horizontal axis (ρ_{16}). The coverage ratio is computed exactly as in Definition 4.1 from π_0 and π_{SFT}: the base-reachable set $\mathcal{R}_0 = \{i : 0.05 < P_{i,16}(\pi_0) < 0.95\}$ uses the *empirical* base pass@16, while L and G use the plug-in $f_{16}\big(P_{i,1}(\cdot)\big)$ of the empirical pass@1. Standard SFT is the no-filter (f=0) run. The size $|\mathcal{R}_0|$ of the base-reachable set varies from 16 (AIME) to 237 (CruxEval-O) and is listed per point in Table 29.

Filter	Level 1 (n=43)	Level 2 (n=90)	Level 3 (n=105)	Level 4 (n=128)	Level 5 (n=134)
no filter	85.27 ± 1.46	76.25 ± 0.43	65.95 ± 1.59	47.02 ± 0.65	24.66 ± 0.63
static f=0.0625	85.66 ± 1.05	76.44 ± 1.15	64.98 ± 1.93	46.92 ± 0.21	25.67 ± 0.72
static f=0.125	84.98 ± 0.89	76.69 ± 0.70	64.46 ± 0.77	47.28 ± 0.47	25.56 ± 0.73
static f=0.1875	84.79 ± 0.75	75.83 ± 1.05	65.06 ± 0.51	46.14 ± 1.65	25.11 ± 0.66
static f=0.25	85.08 ± 0.69	75.62 ± 0.77	65.08 ± 0.34	47.18 ± 0.86	25.51 ± 0.33
static f=0.5	83.58 ± 0.58	74.31 ± 0.90	62.38 ± 1.11	46.48 ± 0.13	24.08 ± 0.03
ramp 0→0.0625	85.61 ± 1.54	76.06 ± 0.51	66.03 ± 0.72	47.27 ± 0.30	25.30 ± 0.56
ramp 0→0.125	84.20 ± 0.67	76.41 ± 1.28	65.24 ± 1.17	47.20 ± 1.10	24.58 ± 0.82
ramp 0→0.1875	85.90 ± 0.52	76.39 ± 0.84	65.85 ± 0.30	48.00 ± 0.09	25.65 ± 1.10
ramp 0→0.25	85.85 ± 1.24	76.44 ± 0.26	65.66 ± 0.53	47.58 ± 0.31	25.37 ± 0.69
ramp 0→0.5	85.27 ± 0.22	75.93 ± 0.54	65.28 ± 0.61	47.84 ± 0.79	24.75 ± 1.02

Table 21: MATH-500 SFT **pass@1** by difficulty level, from the same sweep as Table 20. Values are percent, mean ± sample standard deviation over three seeds.

Eval family	Filter	pass@1	Δ	pass@16	Δ
base (no SFT)†		5.16	—	20.91	—
ramp sweep	no filter	3.65 ± 0.37	—	15.24 ± 2.02	—
ramp sweep	ramp 0→0.1875	3.82 ± 0.20	+0.16	16.06 ± 1.42	+0.82
ramp sweep	ramp 0→0.25	3.78 ± 0.20	+0.13	15.37 ± 0.23	+0.13
ramp sweep	ramp 0→0.5	4.03 ± 0.02	+0.37	18.31 ± 0.57	+3.07
static	no filter	3.65 ± 0.34	—	15.52 ± 2.00	—
static	static f=0.125	3.75 ± 0.18	+0.10	16.14 ± 1.73	+0.63

Table 22: AIME 2022–2025 (n=120) completed SFT filtering evaluations. The ramp sweep and the static pair were evaluated in separate runs, each against its own matched no-filter baseline. Values are percent, mean ± sample standard deviation over three seeds; Δ is the change against the matching no-filter row. †The *base (no SFT)* row is the untuned OLMo-3 7B base under the same AIME protocol.

Vertical axis (coverage gain Δ). The plotted gain is the mean over $\mathcal{R}_0$ of the empirical pass@16 difference $P_{i,16}(\pi_{\text{Tail}}) - P_{i,16}(\pi_{\text{SFT}})$, in percentage points. The TailSFT and standard runs use the same clip settings surfaced in Table 1: static f=0.25 for BigCode and Magicoder, ramp 0→0.5 for OCI, ramp 0→0.5 for AIME, OMEGA-500, and MATH-500 Level 5.

Special cases. Two points require extra care. For *MATH-500 Level 5*, the base model was only evaluated at a 4096-token generation length, so to keep π_0 and π_{SFT} length-matched inside ρ_{16} we compute $\mathcal{R}_0$, L, and G from a 4096-token base run and a 4096-token no-filter run, restricted to the 134 Level-5 problems; the plotted gain Δ still uses the 8192-token standard and TailSFT runs of Table 1. Recomputing ρ_{16} from the 8192-token no-filter run instead leaves it essentially unchanged (2.28 vs. 2.04, both > 1), so the point's placement is robust to this choice. For *OMEGA-500*, per-problem pass@1 and pass@16 are recomputed from the stored generations with the same exact-match (flexible) grader used for Table 1, rather than read from cached metric files, so the diagnostic and the main table score the runs identically. Each point is thus a triple $(\rho_{16}, \Delta, |\mathcal{R}_0|)$; Table 29 lists all eighteen.

SFT data	Benchmark	Standard p@1	TailSFT p@1	Standard p@16	TailSFT p@16
OMI	AIME 2022–2025 ($n = 120$)	$3.65_{\pm 0.37}$	$4.03_{\pm 0.02}$	$15.24_{\pm 2.02}$	$18.31_{\pm 0.57}$
OMI	MATH-500 Level 5 ($n = 134$)	$24.80_{\pm 0.40}$	$25.16_{\pm 0.40}$	$66.42_{\pm 0.00}$	$69.15_{\pm 0.86}$
OMI	OMEGA-500 ($n = 500$)	$6.58_{\pm 0.50}$	$6.51_{\pm 0.26}$	$32.80_{\pm 2.25}$	$32.60_{\pm 1.56}$

Table 23: Post-SFT math results, matching Table 1. Values are percentages, mean ± sample standard deviation over three seeds.

Table 24: SFT pass@k deltas for math benchmarks, reported as TailSFT minus Standard SFT in percentage points at the main-table settings. Deltas are computed with the unbiased OLMES (Gu et al., 2025) pass@k estimator.

Benchmark	Δp@1	Δp@2	Δp@4	Δp@8	Δp@16
AIME 2022–2025 (n=120)	+0.37	+0.72	+1.20	+1.93	+3.07
MATH-500 Level 5 (n=134)	+0.36	+0.71	+0.83	+0.87	+2.74
OMEGA-500 (n=500)	−0.06	−0.09	−0.12	−0.23	−0.20

Setting	Value
Initialization	matched Standard / TailSFT OMI SFT checkpoints (Table 1)
Algorithm	GRPO, group-relative advantage, no KL
Training prompts	MATH-lighteval train, MATH-500 removed (11,396 train / 600 val)
Reward	verifiable boxed-answer exact match (1/0)
Rollout group size n	4
Actor learning rate	searched $\{1, 2\} \times 10^{-5}$; reported 2×10^{-5}
Max prompt / response length	512 / 4096
Epochs	3
Prompt batch / mini-batch	128 / 128
Checkpoint selection	mean-best (highest mean validation reward)
Evaluation	MATH-500 L5 (temp 1.0, 16 samples) & AIME (temp 0.6, 32 samples); 8192-token, 3 seeds

Table 25: Math GRPO configuration. All runs share the fixed settings; only the actor learning rate is searched, and Table 26 reports the tuned configuration.

	pass@1			**pass@16**		
Benchmark	Standard	TailSFT	Δ	Standard	TailSFT	Δ
MATH-500 Level 5 (n=134)	57.70 ± 0.44	60.26 ± 1.02	+2.56	83.83 ± 1.72	87.06 ± 0.86	+3.23
AIME 2022–2025 (n=120)	14.40 ± 0.26	15.61 ± 0.44	+1.21	32.76 ± 0.88	36.06 ± 0.98	+3.30

Table 26: Math GRPO at the tuned configuration (rollout n=4, actor learning rate 2×10^{-5}, mean-best checkpoint, 8192-token generation), evaluated from the matched Standard and TailSFT initializations of Table 1. Values are percent, mean ± standard deviation over three seeds; Δ is the TailSFT improvement.

Setting	Value
Initialization	matched Standard / TailSFT SFT checkpoints per dataset (Table 1)
Algorithm	GRPO, group-relative advantage, no KL
Training prompts	MBPP+ train (100-problem held-out validation)
Reward	full MBPP+ unit-test pass (1/0)
Rollout group size n	searched $\{2, 4\}$; reported 4
Actor learning rate	searched $\{5 \times 10^{-6}, 1 \times 10^{-5}, 2 \times 10^{-5}\}$; reported 2×10^{-5}
Max prompt / response length	1024 / 1024
Epochs	10
Prompt batch / mini-batch	128 / 128
Checkpoint selection	highest validation pass rate
Evaluation	EvalPlus MBPP+ (n=378); 2048-token, temp 1.0, top-p 1.0, 16 samples × 3 seeds

Table 27: Code GRPO configuration. All runs share the fixed settings; the rollout group size and actor learning rate are searched, and Table 28 reports the tuned configuration for each dataset.

			pass@1			**pass@16**		
SFT data (TailSFT filter)	n	LR	Standard	TailSFT	Δ	Standard	TailSFT	Δ
BigCode (static f=0.25)	4	2×10^{-5}	69.57 ± 0.29	73.50 ± 0.70	+3.93	75.93 ± 0.00	78.66 ± 0.15	+2.73
Magicoder (static f=0.25)	2	1×10^{-5}	64.08 ± 0.32	66.02 ± 0.55	+1.94	75.84 ± 0.31	79.28 ± 0.55	+3.44
Magicoder (static f=0.25)	4	2×10^{-5}	70.52 ± 0.25	73.24 ± 0.09	+2.72	78.22 ± 0.61	80.60 ± 0.55	+2.38
OCI (ramp 0→0.5)	4	2×10^{-5}	74.67 ± 0.08	76.30 ± 0.05	+1.62	84.22 ± 0.40	84.04 ± 0.15	−0.18

Table 28: Code GRPO (MBPP+, n=378, 2048-token generation) from the matched Standard and TailSFT initializations of Table 1, at tuned configurations. n is the GRPO rollout group size and LR the actor learning rate. Values are percent, mean ± standard deviation over three seeds; Δ is the TailSFT improvement, with yellow marking a change within one standard deviation.

SFT dataset	Benchmark	$\lvert\mathcal{R}_0\rvert$	ρ_{16}	Δ (pp)
OMI	AIME	16	7.60	+9.92
	MATH-500 L5	31	2.28	+3.23
	OMEGA-500	140	0.90	−0.95
BigCode	CruxEval-I	199	1.63	+3.02
	CruxEval-O	237	2.87	+28.69
	HumanEval+	29	1.39	+9.20
	LiveCodeBench	105	1.37	+3.81
	MBPP+	50	1.30	+14.00
Magicoder	CruxEval-I	199	2.26	+12.40
	CruxEval-O	237	0.73	+18.42
	HumanEval+	29	1.04	−0.00
	LiveCodeBench	105	1.22	+1.59
	MBPP+	50	0.77	+11.33
OCI	CruxEval-I	199	1.35	+6.37
	CruxEval-O	237	0.55	+9.00
	HumanEval+	29	0.23	−6.32
	LiveCodeBench	105	0.97	−1.27
	MBPP+	50	0.54	+2.00

Table 29: Exact values plotted in Figure 4: the base-reachable set size $\lvert\mathcal{R}_0\rvert$, the coverage ratio ρ_{16} (Definition 4.1), and the coverage gain Δ (mean empirical pass@16 difference of TailSFT over standard SFT on $\mathcal{R}_0$, in percentage points), for each (SFT dataset, benchmark) pair.

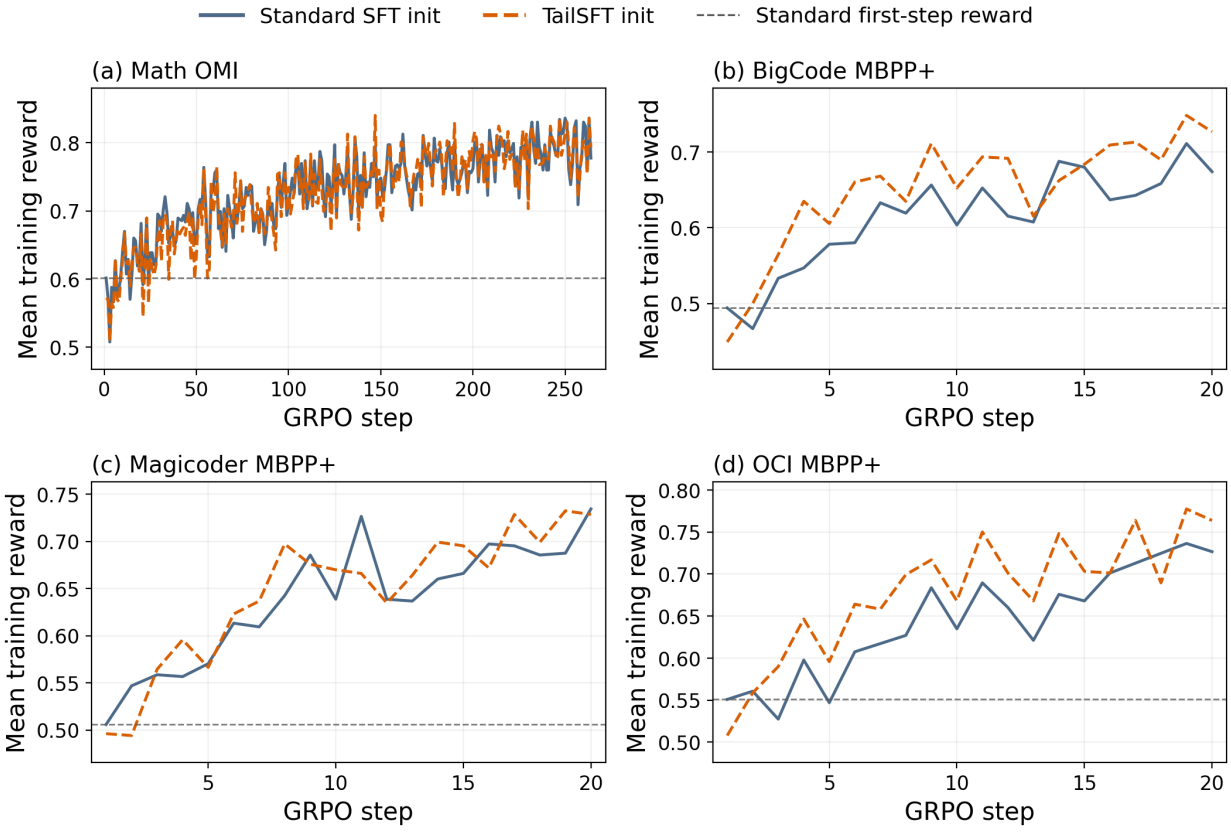

Figure 8: **GRPO training reward from Standard and TailSFT initializations.** Mean training reward (`critic/rewards/mean`) versus GRPO step for the four main-table settings. The horizontal dashed line marks the Standard initialization's first-step reward. TailSFT begins below Standard but recovers Standard's starting reward within a small fraction of training and rises at least as fast thereafter.