# TASK VECTOR DESCENT: LEARNING FROM NON-IID BATCHES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A central challenge in continual learning is to acquire new knowledge without forgetting what the model has already learned. This challenge appears in language model training when training data comes from various domain-, user-, or task-specific distributions that are encountered unevenly over time. In such settings, successive minibatches are temporally clustered by distribution instead of being sampled i.i.d. from the overall data mixture. Training on temporally clustered data induces a stability-plasticity tradeoff. Adapting the model to the active distribution can improve the model on the active distribution but may lead to a performance degradation on data it previously trained on. We find that this tradeoff intensifies with longer exposure to the same distribution. We therefore ask if the parameter displacement produced by such a sequence (the task vector) should be fully retained or applied partially. We compare applying the full displacement ($\lambda = 1$) with partial integration, which scales the task vector by $\lambda$ before applying it to the continuing model and scales the optimizer state by the same coefficient. Across continual pretraining, pretraining from random initialization, supervised post-training, and reinforcement post-training, we find that intermediate values of $\lambda$ often improve average continuing-model performance relative to full integration, particularly after longer same-distribution sequences. In continual-pretraining experiments with both controlled streams and naturally defined adaptation sequences, task-vector scaling outperforms full integration at the matched learning rate, showing that its benefits are not reproduced by learning-rate scaling alone.

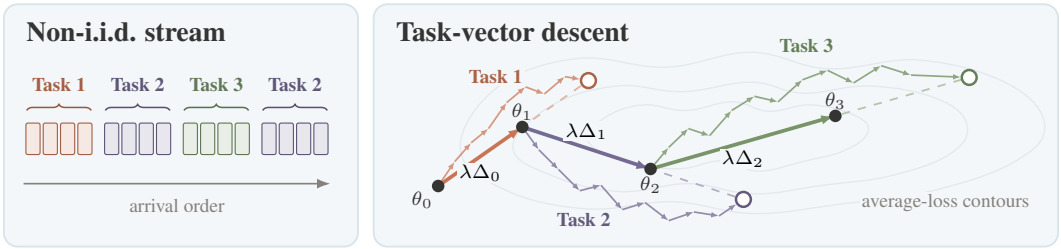


Figure 1: Task-vector updates for non-i.i.d. minibatch streams. Minibatches arrive in task-clustered sequences (left). For each task, noisy minibatch updates produce a local endpoint; the continuing model applies a scaled task vector and begins the next task from the resulting checkpoint (right).

## 1 INTRODUCTION

A long-term goal for language-model agents is to become more capable through experience, i.e., to learn from interactions with users, adapt to new tasks, and carry what they learn into future interactions. Achieving this requires accumulating knowledge and capabilities over time without losing those acquired earlier. The data available for such adaptation comes from multiple domains, sources, users, tasks, or reinforcement-learning environments (Udandarao et al., 2024). These distributions are encountered unevenly over time, with samples from the same distribution often becoming available close together (Cai et al., 2021; Jones et al., 2023), while staged post-training exposes models to successive tasks and objectives, such as supervised fine-tuning followed by preference optimization



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

or reinforcement learning. In conventional offline training, data collected across distributions can be pooled and shuffled to approximate i.i.d. sampling from the overall mixture (Mayo et al., 2023). We instead consider an online continual learner that updates when new data become available rather than waiting for a globally mixed dataset (Hu et al., 2023; Cai et al., 2021). Such a learner, therefore, encounters a temporally clustered, non-i.i.d. stream and must maintain performance across all distributions represented in that stream. A further motivation comes from test-time training, where adaptation can improve performance on a target task (Hübotter et al., 2025). Extending such adaptation to successive tasks raises the question of how much of each update to retain.

When consecutive minibatches come from the same distribution, standard gradient-based training progressively adapts the model to that targeted distribution. Thus, longer sequences can produce stronger adaptation and a larger displacement from the model at the start of the sequence. We refer to this displacement as a *task vector* (Ilharco et al., 2023). Standard continual training uses the model obtained at the end of each sequence as the starting point for the next, retaining the entire parameter displacement. For long sequences, full integration risks overadapting the model to the targeted distribution and degrading performance on non-targeted distributions (Mayo et al., 2023). As this interference accumulates across the stream, it can lead to catastrophic forgetting (McCloskey & Cohen, 1989; Goodfellow et al., 2014).

*How can a language model benefit from sustained adaptation to one distribution while retaining capabilities needed on others?* We study whether full integration remains the most effective update as the minibatch stream moves further from i.i.d. sampling, or whether the task vector should be partially integrated instead. Our contribution is a controlled empirical study of how temporal clustering affects partial integration across pretraining and post-training settings. Holding the selected training inputs fixed, we vary the length of same-distribution sequences and the integration coefficient $\lambda$, and examine their effects on targeted adaptation and continuing-model performance. Specifically, we study a linear family of update rules that moves the continuing model a fraction $\lambda$ along the task vector and applies the same coefficient to the optimizer state. Full integration ($\lambda = 1$) recovers standard continual training, whereas partial integration ($0 < \lambda < 1$) retains only a fraction of the task vector.

This formulation motivates the following **research questions**.

- **RQ1:** Across adaptation objectives and lengths of same-distribution sequences, when does task-vector scaling improve average continuing-model performance relative to full integration, and does it outperform full integration at the matched learning rate $\lambda\eta$?
- **RQ2:** How does $\lambda$ trade targeted-distribution adaptation against non-targeted retention?
- **RQ3:** Does task-vector scaling improve continuing-model performance during pretraining from random initialization, when no shared pretrained representation is available?
- **RQ4:** Does task-vector scaling remain effective when adaptation sequences follow natural data boundaries and vary in length?

Across continual pretraining, pretraining from random initialization, persona adaptation with on-policy self-distillation, supervised fine-tuning (SFT), and Group Relative Policy Optimization (GRPO), intermediate values of $\lambda$ improve the continuing-model performance compared with full integration, particularly after long same-distribution sequences. In continual pretraining, task-vector scaling also outperforms matched learning-rate scaling. The coefficient $\lambda$ trades adaptation to the targeted distribution *(plasticity)* against retention on non-targeted distributions *(stability)*. Improvements during pretraining from random initialization show that this effect does not require a shared pretrained representation. These benefits also extend to book-level continual pretraining with naturally defined, variable-length adaptation sequences. Finally, in our experiments, interpolation remains well-behaved even after extended supervised fine-tuning, whereas pretraining-scale adaptation produces severe performance barriers.

## 2 Related work

**Continual learning on non-i.i.d. data** Continual learning aims to learn from sequentially arriving data without forgetting earlier capabilities. Sequential learning is also studied through regret minimization in online convex optimization (Zinkevich, 2003; Hazan, 2016) and lifelong bandit optimization (Schur et al., 2023). These works analyze cumulative performance on encountered tasks, while continual learning additionally emphasizes retaining performance on earlier tasks. Long

sequences of gradient updates on one distribution can overadapt the model and cause catastrophic forgetting (McCloskey & Cohen, 1989; Goodfellow et al., 2014; De Lange et al., 2022). Prior work addresses forgetting through replay, regularization, and task-specific parameter isolation (De Lange et al., 2022) and studies forgetting in pretrained representations, language-model tuning, and continual pretraining (Luo et al., 2023; Li et al., 2024; Udandarao et al., 2024). Complementing this line of work, we study the effect of temporal clustering on continual training by holding the samples fixed across conditions and varying only their order, thereby changing how many consecutive minibatches come from the same distribution.

**Model merging, task arithmetic, and mode connectivity** Model merging combines checkpoints adapted separately to different tasks or domains: Fisher-weighted averaging accounts for parameter importance, task arithmetic represents fine-tuning as a displacement from the shared initialization, and TIES-merging addresses interference between task-specific updates (Matena & Raffel, 2022; Ilharco et al., 2023; Yadav et al., 2023). Even when two checkpoints perform well individually, models along the straight line between their parameters may perform poorly. Mode-connectivity work shows that such checkpoints can often be connected by curved paths along which the loss remains low (Garipov et al., 2018; Draxler et al., 2018). Linear connectivity is more likely when checkpoints share optimization history or parameter symmetries are aligned (Frankle et al., 2020; Vlaar & Frankle, 2022; Entezari et al., 2022; Ainsworth et al., 2023). Because each locally adapted endpoint descends from the corresponding continuing-model parameters, these results motivate task-vector integration while also raising the question of when it ceases to be effective.

**Continual model merging** Lookahead performs several inner-optimizer steps on fast weights before moving slow weights partway toward them, yielding the same parameter-interpolation form studied here. It also evaluates retaining, interpolating, or resetting SGD momentum (Zhang et al., 2019). Sequential Fine-tuning with Averaging (SFA) periodically averages the adapted model with the checkpoint preceding the current task and continues from the merged model (Kleiman et al., 2025). AIMMerging determines when to merge using parameter displacement and replay loss, and combines task vectors obtained from task-specific and replay data (Feng et al., 2025). Perturb-and-Merge uses a dynamic merging coefficient derived from empirical Fisher information and a curvature-regularized adaptation objective (Qiu et al., 2025). We study how the benefits of partial integration depend on temporal clustering in language-model training, varying the length of same-distribution sequences while holding the selected training inputs fixed. An extended discussion of related work is provided in Appendix A.

## 3 Task-vector updates for non-i.i.d. minibatch streams

### 3.1 Task-vector updates

We separate adaptation to the current distribution from the amount of parameter change carried into subsequent learning. This allows the model to follow the local optimization trajectory at the original learning rate while controlling how far the continuing model moves toward the locally adapted endpoint. At sequence boundary $t$, the continuing checkpoint is the pair $(\theta_t, \omega_t)$, comprising the model parameters and optimizer state. We call the model parameterized by $\theta_t$ the *continuing model*.

**Learning objective** Let $P_1, \ldots, P_K$ be fixed data distributions, each corresponding to a domain, user, task, or environment, and let $\mathcal{L}(\theta; z)$ denote the loss on a sample $z$. Our goal is to minimize the loss averaged uniformly across these distributions

$$\min_{\theta} \mathbb{E}_k \mathbb{E}_{z \sim P_k} \left[ \mathcal{L}(\theta; z) \right]. \tag{1}$$

For a given distribution $P_k$, training estimates its population loss from a minibatch $B$ containing i.i.d. samples from $P_k$, using the empirical loss $\mathcal{L}(\theta; B) = \frac{1}{|B|} \sum_{z \in B} \mathcal{L}(\theta; z)$. We consider $L$ consecutive minibatches $B_1, \ldots, B_L$ drawn from the same distribution.

**Minibatch updates** For clarity, we illustrate local adaptation using SGD without momentum. Starting from $\theta_t$, we process the next $L$ minibatches with learning rate $\eta$:

$$\theta_t^{(0)} = \theta_t, \qquad \theta_t^{(i)} = \theta_t^{(i-1)} - \eta \nabla_\theta \mathcal{L}\left(\theta_t^{(i-1)}; B_i\right), \qquad i = 1, \ldots, L. \tag{2}$$

We refer to these $L$ minibatch updates as *local adaptation* and to $\theta_t^{(L)}$ as the *locally adapted endpoint*. Our experiments use AdamW, which updates both the model parameters and optimizer state. The task-vector definition below applies to the resulting parameter displacement regardless of the optimizer used for local adaptation.

**Task-vector update** The displacement produced by $L$ minibatch updates defines the task vector $\Delta_t$. We construct the next continuing-model parameters by taking a step of size $\lambda$ along this vector:

$$\Delta_t = \theta_t^{(L)} - \theta_t^{(0)}, \qquad \theta_{t+1} = \theta_t + \lambda \Delta_t, \qquad \lambda \in [0, 1]. \tag{3}$$

We call the operation that forms $\theta_{t+1}$ from $\theta_t$ and $\Delta_t$ *integration*. Full integration uses $\lambda = 1$ and, together with full optimizer-state integration, recovers standard training with the chosen optimizer, while partial integration uses $0 < \lambda < 1$. Setting $\lambda = 0$ discards the task vector. We refer to this family of scaled updates as *Task Vector Descent*.

**Optimizer-state integration** Our experiments use AdamW (Loshchilov & Hutter, 2019). We interpolate the optimizer state alongside the parameters. For the first moment, this reduces the current sequence's contribution while preserving more of the inherited momentum. At $\lambda = 0$, we restore the pre-sequence parameters and optimizer state; at $\lambda = 1$, we recover standard training. Let $\omega_t = (m_t, v_t, c_t)$ denote the optimizer state associated with $\theta_t$, comprising the first moment, second moment, and step count, and let $\omega_t^{(L)} = (m_t^{(L)}, v_t^{(L)}, c_t^{(L)})$ denote the state after the $L$ minibatch updates. We integrate these states using the same coefficient $\lambda$:

$$m_{t+1} = (1-\lambda) m_t + \lambda m_t^{(L)}, \quad v_{t+1} = \left( (1-\lambda)\sqrt{v_t} + \lambda \sqrt{v_t^{(L)}} \right)^2. \tag{4}$$

The first moment and step count are interpolated linearly, while the second moment is interpolated in square-root space because AdamW's adaptive normalization depends on the root-mean-square gradient scale rather than the squared scale. Together, $(\theta_{t+1}, \omega_{t+1})$ form the continuing checkpoint passed to the next sequence. The complete training procedure, AdamW updates, and optimizer-state integration details are provided in Appendix C.

**Understanding partial integration** Our analysis clarifies why task-vector scaling differs from reducing the learning rate and when partial integration can improve on full integration. A second-order SGD analysis shows that both methods agree to first order, but task-vector scaling retains a larger contribution from second-order interactions within each adaptation sequence (Appendices B.1 and B.2). A local quadratic model favors partial integration when the full update overshoots the average-loss minimum along the task vector (Appendix B.3).

### 3.2 Controlled stream construction

For each distribution $P_k$, we use disjoint training and evaluation data. The controlled streams contain only training examples, while the evaluation data are used solely to compute the scores defined below.

We construct controlled non-i.i.d. streams by fixing the minibatches drawn from each distribution and varying only their order. For a chosen $L$, we partition the minibatches associated with each $P_k$ into sequences of $L$ consecutive minibatches and shuffle the order of these sequences. Let $k_t$ index the distribution supplying sequence $t$. We call $P_{k_t}$ the targeted distribution and all $P_k$ with $k \neq k_t$ the non-targeted distributions. Minibatches are consumed without replacement and the stream is balanced across distributions. After each sequence, we apply the task-vector update defined in Equation (3). For fixed distributions and sampling frequencies, $L$ controls temporal clustering and integration frequency: larger $L$ produces longer same-distribution sequences, whereas smaller $L$ brings the stream closer to i.i.d. sampling. Even when $L = 1$, however, each minibatch contains samples from only one distribution and is not fully i.i.d. We thus construct a separate i.i.d. reference that mixes distributions within each minibatch. We hold the samples and training budget fixed across conditions and vary only their order. Fig. 9 in the appendix illustrates this construction.

### 3.3 Evaluation

At each evaluated sequence boundary, we score both the locally adapted endpoint $\theta_t^{(L)}$ and the continuing model $\theta_{t+1}$ to distinguish local adaptation from performance retained for subsequent

learning. The endpoint $\theta_t^{(L)}$ measures performance on the targeted distribution after local adaptation, while $\theta_{t+1}$ measures performance carried forward after integration. Comparing their scores isolates the immediate effect of scaling at that boundary, with both checkpoints sharing the same preceding training history. These evaluations define continuing-model gain, stability and plasticity, and the immediate scaling effect, as summarized in Fig. 2.

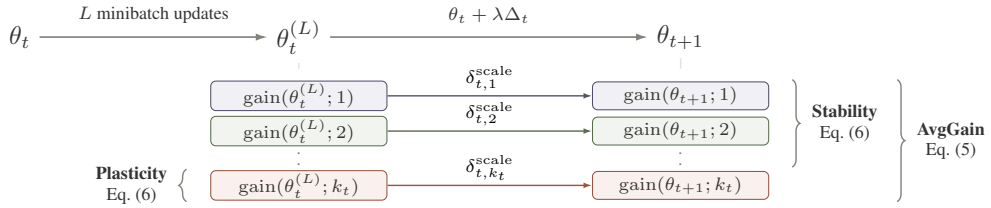


Figure 2: Sequence-boundary evaluation. Plasticity uses the locally adapted endpoint; stability and AvgGain use the continuing model.

Let $\text{score}(\theta; k)$ denote the higher-is-better held-out score of the model with parameters $\theta$ on distribution $k$. For language modeling, we use negative bits per token (BPT) as the higher-is-better score. We define its gain relative to the initial model $\theta_0$ as $\text{gain}(\theta; k) = \text{score}(\theta; k) - \text{score}(\theta_0; k)$.

**Average gain (AvgGain)** We use average gain to measure improvement across all distributions after integration. Intuitively, it asks whether the model becomes better overall as it learns from successive distributions. At sequence boundary $t$, the average gain of the continuing model after integration is

$$\text{AvgGain}_t = \frac{1}{K}\sum_{k=1}^{K} \text{gain}(\theta_{t+1}; k). \tag{5}$$

**Stability and plasticity** However, average gain can conceal a trade-off between adapting to the current distribution and maintaining performance elsewhere. At sequence boundary $t$, we therefore measure plasticity as the targeted-distribution gain at the locally adapted endpoint and stability as the mean non-targeted gain after integration:

$$\text{Plasticity}_t = \text{gain}\left(\theta_t^{(L)}; k_t\right), \qquad \text{Stability}_t = \frac{1}{K-1}\sum_{k \neq k_t} \text{gain}\left(\theta_{t+1}; k\right). \tag{6}$$

We report these quantities averaged over evaluated sequence boundaries and seeds. Although plasticity is measured before scaling, it still depends on $\lambda$ through the model produced by previous steps.

## 4 Experimental setup and applications

We choose settings covering the main stages of LLM training to test whether the benefits of partial integration extend across different initializations and learning objectives. Complete training configurations, evaluation protocols, and interpolation setups are provided in Appendix D.

**Common setup** Except for book-level pretraining, selected data are fixed across seeds; seeds control distribution order, training randomness, and stochastic generation. Each run uses a fixed integration coefficient $\lambda$. We average scores over examples, tasks, and evaluated integration steps, then report the mean and standard error of the mean (SEM) across seeds. A training unit is an optimizer minibatch for pretraining and SFT, a prompt–persona interaction for persona adaptation, or a four-prompt rollout batch for GRPO. Table 1 in the appendix summarizes the experimental settings and metrics.

**Continual pretraining** A continually improving language model must incorporate knowledge from newly available sources without losing its usefulness on others. We study this balance through continual pretraining on temporally clustered domains. We continue training the released 124M-parameter GPT-2 model (Radford et al., 2019) on ten Pile Uncopyrighted domains (Gao et al., 2020), using document-disjoint training and evaluation splits. We select the learning rate $\eta = 4 \times 10^{-5}$ through a grid search on the i.i.d. reference. As a matched learning-rate baseline, we use full integration with learning rate $\lambda\eta$. The primary score is BPT reduction relative to the initial checkpoint.

**Persona adaptation** Persona adaptation tests whether a model can learn a user's preferences while retaining performance for other users. We adapt a Qwen3-4B student to six style preferences using

Qwen3-8B as simulator and judge (Yang et al., 2025; Kleine Buening et al., 2026). Each preference uses the same 32 HelpSteer prompts (Wang et al., 2024b). Rejected interactions provide simulator feedback and trigger Self-Distillation Policy Optimization (SDPO) updates (Hübotter et al., 2026), while accepted interactions are skipped. SDPO uses the student's feedback-conditioned predictions to teach the same student without feedback. The primary score is macro-average pairwise win rate against the initial student on 128 held-out prompts.

**Reasoning SFT and GRPO** Learning to solve new reasoning tasks should preserve the ability to solve previously encountered ones. We use SFT and GRPO to test whether partial integration improves this balance under supervised and reward-based learning objectives. We adapt Qwen2.5-3B-Instruct (Yang et al., 2024) to Countdown, Mini-Sudoku, N-Queens, and String Manipulation from Reasoning Gym (Stojanovski et al., 2025). SFT uses completion-only cross-entropy on 256 verified expert traces per task; GRPO uses verifiable rewards on 4,096 prompts per task (Shao et al., 2024; Guo et al., 2025). We evaluate 100 prompts per task with four sampled responses, using Best@4 as the primary score.

**Pretraining from random initialization** To test whether partial integration is effective without a pretrained representation, we train a randomly initialized 124M-parameter GPT-2-size decoder-only Transformer in BF16 on approximately 10.77B tokens from ten Pile Uncopyrighted domains (Gao et al., 2020). Each effective minibatch contains 512 packed 1,028-token blocks. The i.i.d. reference uses globally shuffled mixed-domain minibatches.

**Book-level continual pretraining** Books provide natural sequence boundaries for testing adaptation beyond our controlled streams. We continually adapt SmolLM-360M (Ben Allal et al., 2024) on 200 PG-19 books (Rae et al., 2020), treating each book as one sequence and integrating after each book. We compare partial integration with full integration at matched learning rates $\lambda\eta$, using identical training manifests within each of three paired seeds. The primary score is BPT reduction relative to initialization on 50 fixed validation books, averaged equally over books and the 200 integration boundaries.

**Controlled long-SFT interpolation** To test interpolation after extended adaptation, we interpolate SmolLM2-1.7B-Instruct with checkpoints along a MetaMathQA SFT trajectory and evaluate GSM8K accuracy. We also examine pretraining-scale interpolation between Qwen2.5 base models and their Qwen2.5-Coder descendants.

# 5 Results

## 5.1 Intermediate task-vector scaling improves the continuing model

To address **RQ1**, we test whether partial integration improves performance across distributions and whether its benefits can be explained by smaller optimization steps. We measure the cumulative effect of the update rule through $\mathrm{AvgGain}_t$ (Eq. (5)), averaged over the evaluated sequence boundaries. This captures both the immediate effect of each integration and its influence on the continuing checkpoints inherited by later sequences.

**Continual pretraining** Task-vector scaling improves average gain, with the benefits becoming more pronounced for longer same-distribution sequences (Fig. 3). For every evaluated $L \geq 32$, an intermediate coefficient performs best, and task-vector scaling consistently outperforms matched learning-rate scaling for every $\lambda < 1$. At $L = 512$, for example, $\lambda = 0.6$ increases the BPT gain from $0.386$ under full integration to $0.463$, whereas full integration at the matched learning rate $0.6\eta$ reaches only $0.412$. This is consistent with the distinction between task-vector scaling and learning-rate scaling discussed in Section 3. For $L = 1$, by contrast, the two scaling methods perform nearly identically, and full integration performs best and nearly matches the i.i.d. reference.

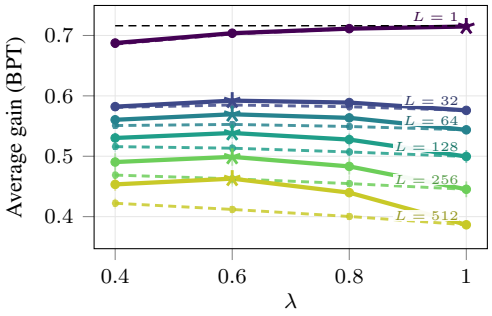


Figure 3: Continual-pretraining gain. Solid curves show task-vector scaling; dashed curves show full integration at learning rate $\lambda\eta$. Stars mark the best task-vector coefficient, error bars show SEM, and the black dashed line denotes the i.i.d. reference.

**Post-training** We next examine whether the benefit extends beyond next-token pretraining to supervised and feedback-based adaptation. Across SDPO, SFT, and GRPO, intermediate coefficients outperform $\lambda = 1$ for every evaluated sequence length (Fig. 4). For reasoning SFT with $L = 32$, $\lambda = 0.4$ increases the Best@4 gain from 30.43 to 41.53 percentage points. Both evaluated GRPO sequence lengths attain their largest average gain at $\lambda = 0.6$, whereas the best coefficient for SDPO is $\lambda = 0.4$ at $L = 4$ and $\lambda = 0.2$ at $L \in \{8, 16\}$.

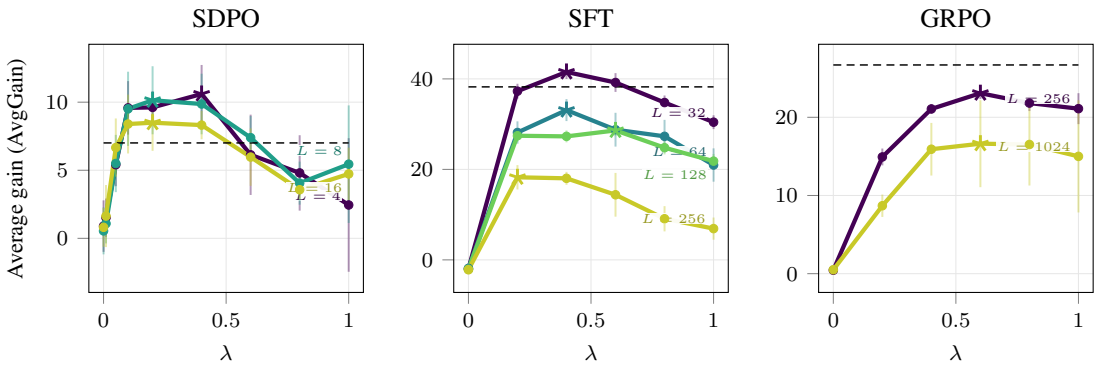


Figure 4: Continuing-model gain for SDPO, SFT, and GRPO (left to right) as a function of $\lambda$. Curves denote $L$; error bars show SEM across seeds, and stars mark the best $\lambda$ for each $L$. Dashed lines are i.i.d. or joint-data references evaluated on the smallest-$L$ schedule in each setting.

Overall, intermediate task-vector scaling is most consistently beneficial after longer sequences from the same distribution, but no single value of $\lambda$ is optimal across adaptation regimes and sequence lengths. Comparisons of $\lambda$ within a fixed $L$ are direct; absolute results across different values of $L$ are not directly comparable because they are evaluated at different sequence boundaries and frequencies.

### 5.2 Task-vector scaling controls the stability–plasticity trade-off

Improved average performance can conceal different effects on adaptation and retention. To address **RQ2**, we examine how task-vector scaling affects learning from the targeted distribution and retaining performance on non-targeted distributions. Separating the locally adapted endpoint from the continuing model is particularly useful in settings such as test-time training: task-specific inference can use the locally adapted endpoint, while only a scaled task vector is retained by the continuing model. We therefore use the stability and plasticity measures defined in Eq. (6) and illustrated in Fig. 2.

**Continual pretraining** Within task-vector scaling, increasing $\lambda$ generally raises plasticity at the cost of stability for $L \in \{32, 64, 128\}$ (Fig. 5). At $L = 256$, the frontier begins to flatten, and at $L = 512$, smaller coefficients improve both coordinates. Task-vector scaling also achieves higher stability and plasticity than matched learning-rate scaling for every sequence length shown, with the gap increasing for longer sequences. At $L = 512$, reducing the learning rate recovers some stability by sacrificing plasticity, whereas partial integration improves both relative to full integration at $\eta$. Because plasticity is measured before the current task vector is scaled, its improvement within task-vector scaling reflects the cumulative effect of $\lambda$ on checkpoints inherited from earlier sequences.

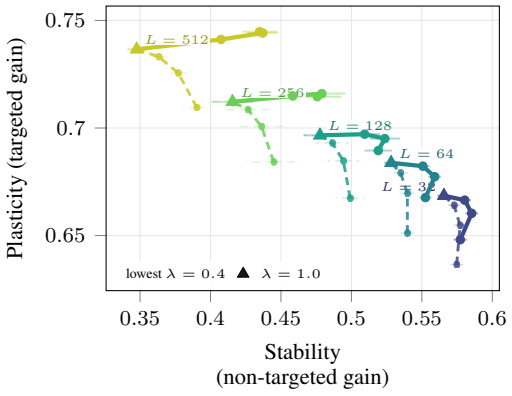


Figure 5: Stability–plasticity trade-off in continual pretraining. Solid curves show task-vector scaling and dashed curves show full integration at learning rate $\lambda\eta$; error bars show SEM.

**Post-training** The post-training settings show the same regime-dependent transition (Fig. 6). For shorter SFT and GRPO sequences, larger

coefficients generally improve plasticity, while intermediate coefficients preserve more stability. With longer sequences, the locally adapted endpoint already reflects substantial within-sequence adaptation; stronger integration then reduces stability and can also impair plasticity at later sequences. This reversal is clearest for SFT with $L = 256$ and GRPO with $L = 1024$. SDPO is less monotonic: because its data are collected on-policy, some intermediate coefficients improve targeted and non-targeted style scores simultaneously.

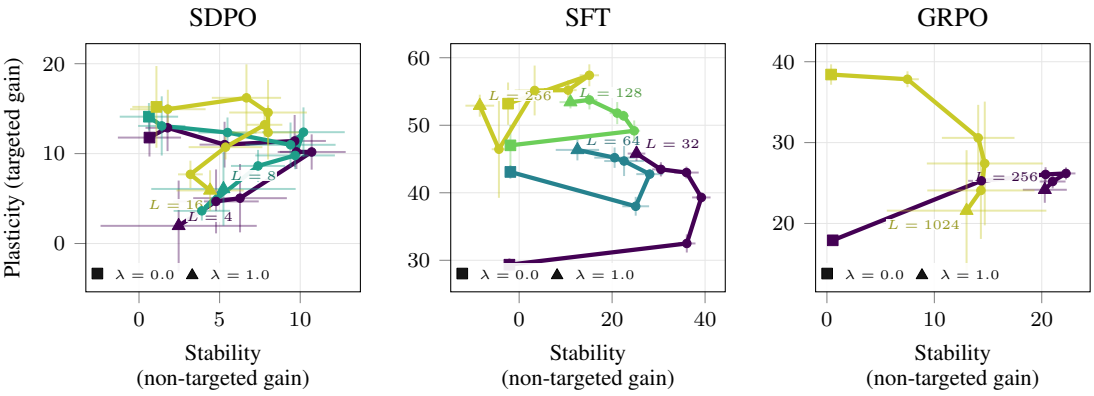


Figure 6: Non-targeted retention versus targeted adaptation for SDPO, SFT, and GRPO from left to right across task-vector scaling coefficients. The stability axis is the mean continuing-model gain over non-targeted distributions or contexts; the plasticity axis is targeted gain at the endpoint $\theta_t^{(L)}$.

Overall, $\lambda$ has both immediate and cumulative effects. It directly trades targeted-distribution adaptation against non-targeted retention at the current sequence boundary. By changing the checkpoint used to initialize subsequent sequences, it also affects their later adaptation and retention, particularly when same-distribution sequences are long. To isolate the immediate effect of task-vector scaling, we compare performance before and after integration. We find that scaling recovers substantial non-targeted performance in SFT but little in GRPO, consistent with reduced forgetting under on-policy methods reported by *RL's Razor* (Shenfeld et al., 2026). Further analysis is provided in Appendix E.1.

### 5.3 Task-vector scaling improves pretraining from random initialization

The preceding experiments leave open whether the benefits of partial integration depend on preserving a useful pretrained representation. To address **RQ3**, we test task-vector scaling during pretraining from random initialization.

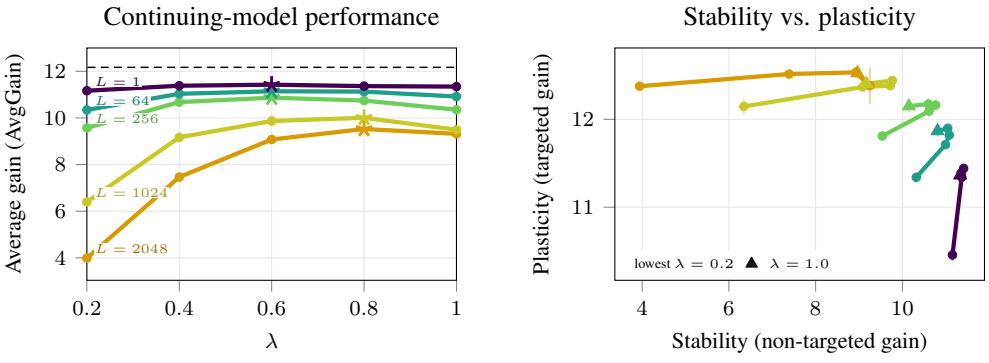


Figure 7: Pre-training from random initialization. The left panel reports continuing-model macro BPT gain as a function of the task-vector scaling coefficient. Stars mark the best coefficient on each curve, and the dashed line is the shuffled i.i.d. reference. The right panel compares plasticity and stability.

Figure 7 shows that an intermediate coefficient improves average continuing-model gain relative to full integration for every evaluated $L$. The best coefficient is $\lambda = 0.6$ for $L \in \{1, 64, 256\}$ and

$\lambda = 0.8$ for $L \in \{1024, 2048\}$. Thus, the benefit of task-vector scaling does not depend on starting from a shared pretrained representation.

The right panel shows that the improvement at the preferred coefficients is driven primarily by higher non-targeted gain, while targeted-distribution gain changes comparatively little. The shuffled i.i.d. condition remains a reference and performs better than all non-i.i.d. configurations. Additional analysis of the immediate effect of scaling is provided in Appendix E.1.

### 5.4 Task-vector scaling improves adaptation with natural sequence boundaries

The previous experiments use equal-length adaptation sequences drawn from a fixed set of recurring tasks. To address **RQ4**, we continually pretrain on a sequence of books, integrating the task vector after training on each book. Book boundaries define the adaptation sequences, whose lengths vary with the number of training minibatches per book. The distribution of token counts per book is reported in Appendix D.8.

Figure 8 shows that partial integration outperforms full integration at the matched learning rate $\lambda\eta$ in all 12 evaluated comparisons with $\lambda < 1$. Using $\eta = 4 \times 10^{-5}$ and $\lambda = 0.4$ yields 12.1% greater average BPT gain than full integration at its best evaluated learning rate, $\eta = 1.6 \times 10^{-5}$. The preferred scaling coefficient depends on the base learning rate: at $\eta = 10^{-5}$, below the best full-integration rate, partial integration reduces gain, whereas it improves gain at the three higher base learning rates. This pattern is consistent with our local analysis, which favors partial integration when the full update overshoots the minimum along the task-vector direction (Appendix B.3).

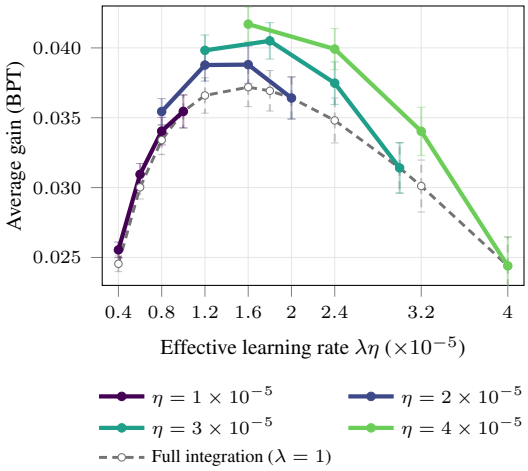


Figure 8: PG-19 BPT gain relative to initialization. Colors denote base learning rates $\eta$; points show $\lambda \in \{0.4, 0.6, 0.8, 1\}$. Gray dashed: full integration at matched effective rates. Error bars: $\pm 1$ SEM over three paired seeds.

### 5.5 Linear interpolation remains effective through long SFT

To probe the limits of task-vector scaling, we test whether linear interpolation remains effective after extended adaptation. We find no pronounced performance barriers along a controlled SFT trajectory, but severe barriers after pretraining-scale adaptation. Experimental details and interpolation curves are provided in Appendices E.2 and E.3.

## 6 Conclusion and limitations

We study task-vector scaling on temporally clustered, non-i.i.d. minibatch streams. Across five training regimes, partial integration often outperforms full integration, especially after longer same-distribution sequences. In domain- and book-level continual pretraining, it also outperforms matched learning-rate scaling. The coefficient controls targeted-distribution adaptation versus non-targeted retention, and the benefits persist during pretraining from random initialization. Direct linear interpolation remains effective throughout our controlled long-SFT trajectory but can fail after pretraining-scale adaptation.

Our experiments cover limited models, data orders, and adaptation regimes. They assume each minibatch is i.i.d. within one distribution and use a fixed integration coefficient $\lambda$. Broader streams and adaptive coefficient selection remain future work.

## Reproducibility Statement

Complete descriptions of the method, training objectives, model and optimizer configurations, stream construction, evaluation procedures, and random seeds are provided in Appendices C and D. We intend to release the code and experiment configurations required to reproduce the results.

## Statement on the use of AI

In this paper, LLMs were used for minor grammatical edits, word polishing, or rephrasing, as well as for assistance with the implementation of the proposed method and experimental code. All suggestions from LLMs were manually verified and edited by the authors prior to final inclusion.

Further, LLMs were used as components of the experimental methodology. In the persona-adaptation experiments, we used Qwen3-8B as a user simulator during training and as a pairwise judge during evaluation. In the reasoning SFT experiments, Claude Sonnet 4.6 was used to generate fixed reasoning traces and answers for the training examples.

## References

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. In *ICLR*, 2023.

Anthropic. Introducing Claude Sonnet 4.6, February 2026.

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

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. SmolLM—blazingly fast and remarkably powerful, 2024.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Clémentine Fourrier, Hugo Larcher, Ben Burtenshaw, Haojun Zhao, Caleb Fahlgren, Mathieu Morlon, Agustín Piqueres Lajarín, Xuan-Son Nguyen, Cyril Zakka, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When smol goes big—data-centric training of a fully open small language model. In *COLM*, 2025.

Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *ICCV*, 2021.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory G. Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(7):3366–3385, 2022.

Thang Doan, Mehdi Abbana Bennani, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Alquier. A theoretical analysis of catastrophic forgetting through the NTK overlap matrix. In *AISTATS*, 2021.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, 2018.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *ICLR*, 2022.

Yujie Feng, Jian Li, Xiaoyu Dong, Pengfei Xu, Xiaohui Zhou, Yujia Zhang, Zexin Lu, Yasha Wang, Alan Zhao, Xu Chu, and Xiao-Ming Wu. AIMMerging: Adaptive iterative model merging using training trajectories for language model continual learning. In *EMNLP*, 2025.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *NeurIPS*, 2018.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *ICLR*, 2014.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2 (3–4):157–325, 2016. doi: 10.1561/2400000013.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021.

Hexiang Hu, Ozan Sener, Fei Sha, and Vladlen Koltun. Drinking from a firehose: Continual learning with web-scale natural language. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(5):5684–5696, 2023.

Jonas Hübotter, Leander Diaz-Bone, Ido Hakimi, Andreas Krause, and Moritz Hardt. Learning on the job: Test-time curricula for targeted reinforcement learning. *arXiv preprint arXiv:2510.04786*.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *ICML*, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*.

Mike Huisman, Aske Plaat, and Jan N. van Rijn. Understanding transfer learning and gradient-based meta-learning techniques. *Mach. Learn.*, 113(7):4113–4132, 2024.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *ICLR*, 2023.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018.

Matt Jones, Tyler R. Scott, Mengye Ren, Gamaleldin Fathy Elsayed, Katherine L. Hermann, David Mayo, and Michael C. Mozer. Learning in temporally structured environments. In *ICLR*, 2023.

Anat Kleiman, Gintare Karolina Dziugaite, Jonathan Frankle, Sham Kakade, and Mansheej Paul. Soup to go: Mitigating forgetting during continual learning with model averaging. *arXiv preprint arXiv:2501.05559*.

Thomas Kleine Buening, Jonas Hübotter, Barna Pásztor, Idan Shenfeld, Giorgia Ramponi, and Andreas Krause. Aligning language models from user interactions. In *COLM*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

Sebastian Lee, Sebastian Goldt, and Andrew Saxe. Continual learning in the teacher–student setup: Impact of task similarity. In *ICML*, 2021.

Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. Revisiting catastrophic forgetting in large language model tuning. In *Findings of EMNLP*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Yun Luo, Zhen Yang, Xuefeng Bai, Fandong Meng, Jie Zhou, and Yue Zhang. Investigating forgetting in pre-trained representations through continual learning. *arXiv preprint arXiv:2305.05968*.

Michael Matena and Colin Raffel. Merging models with Fisher-weighted averaging. In *NeurIPS*, 2022.

David Mayo, Tyler R. Scott, Mengye Ren, Gamaleldin Elsayed, Katherine L. Hermann, Matt Jones, and Michael C. Mozer. Multitask learning via interleaving: A neural network investigation. In *CogSci*, 2023.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. In *NeurIPS*, 2020.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Netw.*, 113:54–71, 2019.

Haomiao Qiu, Miao Zhang, Ziyue Qiao, and Liqiang Nie. Train with perturbation, infer after merging: A two-stage framework for continual learning. In *NeurIPS*, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *ICLR*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *AAAI*, 2020.

Felix Schur, Parnian Kassraie, Jonas Rothfuss, and Andreas Krause. Lifelong bandit optimization: No prior and no regret. In *UAI*, 2023.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In *EMNLP*, 2022.

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 preprint arXiv:2402.03300*.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's razor: Why online reinforcement learning forgets less. In *ICLR*, 2026.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning Gym: Reasoning environments for reinforcement learning with verifiable rewards. In *NeurIPS*, 2025. Datasets and Benchmarks Track.

Vishaal Udandarao, Karsten Roth, Sebastian Dziadzio, Ameya Prabhu, Mehdi Cherti, Oriol Vinyals, Olivier Hénaff, Samuel Albanie, Zeynep Akata, and Matthias Bethge. A practitioner's guide to real-world continual multimodal pretraining. In *NeurIPS*, 2024. Datasets and Benchmarks Track.

Tiffany J. Vlaar and Jonathan Frankle. What can linear interpolation of neural network loss landscapes tell us? In *ICML*, 2022.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers reinforcement learning, 2020. Software.

Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S. Dhillon, and Sanjiv Kumar. Two-stage LLM fine-tuning with less specialization and more generalization. In *ICLR*, 2024a.

Zhilin Wang, Yi Dong, Jiaqi Zeng, Virginia Adams, Makesh Narsimhan Sreedhar, Daniel Egert, Olivier Delalleau, Jane Scowcroft, Neel Kant, Aidan Swope, and Oleksii Kuchaiev. HelpSteer: Multi-attribute helpfulness dataset for SteerLM. In *NAACL*, 2024b.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *EMNLP (System Demonstrations)*, 2020.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022a.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *CVPR*, 2022b.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *NeurIPS*, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are Super Mario: Absorbing abilities from homologous models as a free lunch. In *ICML*, 2024a.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. In *ICLR*, 2024b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *NeurIPS*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *ACL*, 2019.

Michael R. Zhang, James Lucas, Jimmy Ba, and Geoffrey E. Hinton. Lookahead optimizer: $k$ steps forward, 1 step back. In *NeurIPS*, 2019.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

# APPENDIX

The appendix is organized as follows. Appendix A provides an extended discussion of related work, and Appendix B analyzes task-vector updates. Appendices C and D give the complete method and experimental specifications, and Appendix E provides additional results and interpolation diagnostics.

## A EXTENDED RELATED WORK

**Continual learning and forgetting** The goal of continual learning (CL) is to enable a model to accumulate knowledge from a sequence of tasks while preserving capabilities that were obtained previously. While in conventional training data from all tasks are assumed to be jointly available, in CL the model is presented with a sequence or stream of task-specific data distributions and is commonly restricted to data from the current task. A CL method, therefore, must trade off between *plasticity*, the ability to acquire new capabilities, and *stability*, the ability to retain existing capabilities. Adaptation to a new distribution often degrades performance on previously learned tasks. This phenomenon is known as *catastrophic forgetting* (Parisi et al., 2019; De Lange et al., 2022).

Catastrophic forgetting is commonly explained by interference caused through sharing model parameters for multiple tasks. When different tasks rely on overlapping parameters or representations, updates that improve performance on the current task are likely to modify parameters that contribute to performance on previously learned tasks (McCloskey & Cohen, 1989; Goodfellow et al., 2014). The severity of forgetting also depends on the relationship between successive tasks and the alignment of their induced updates (Doan et al., 2021; Lee et al., 2021). Finally, optimization choices such as the learning rate and batch size can substantially affect the model's sensitivity to subsequent updates (Mirzadeh et al., 2020).

Methods for mitigating catastrophic forgetting can be broadly grouped into replay, regularization, and parameter-isolation methods. Replay-based methods keep a subset of previously observed data points and interleave them with data from the current task, providing a direct learning signal for preserving previously obtained capabilities. Regularization-based methods constrain the model to preserve information important for previous tasks, either by penalizing changes to important parameters or by constraining outputs or hidden representations. Parameter-isolation methods reduce interference by assigning separate parameters to different tasks, but may require knowledge of the active task at inference time or cause the number of parameters to grow with the number of tasks (De Lange et al., 2022).

Unlike models trained from scratch on a sequence of tasks, pretrained language models begin with general capabilities acquired from initial training on a large and diverse text dataset. Sequential adaptation can therefore cause two forms of degradation: the model may forget task-specific capabilities acquired during earlier adaptation stages (Scialom et al., 2022), and it may lose general capabilities inherited from pretraining (Luo et al., 2023). Pretraining provides representations that can be reused across tasks and can facilitate transfer during subsequent adaptation. However, fine-tuning on narrow or non-i.i.d. data distributions can adapt the model to the current task at the expense of its broader capabilities (Wang et al., 2024a; Li et al., 2024).

**Weight averaging as a useful operation in weight space** Early work on weight-space averaging showed that linear operations on neural network weights can improve generalization. Stochastic Weight Averaging (SWA) averages checkpoints along a single stochastic gradient descent (SGD) trajectory, resulting in optima in wider regions of the loss landscape that generalize well (Izmailov et al., 2018). While SWA is not model merging in the multi-task sense, it provides early evidence that weight-space averaging can be beneficial when checkpoints are sufficiently aligned.

A related line of work studies averaging of checkpoints from the same base model for robust fine-tuning. WiSE-FT interpolates between a pretrained zero-shot model and its fine-tuned checkpoint to trade off in-distribution (ID) adaptation and robustness under distribution shift (Wortsman et al., 2022b). Model Soups extend this idea by averaging multiple fine-tuned checkpoints, typically obtained from different sets of hyperparameters, improving both ID accuracy and robustness to distribution shift without increasing inference cost (Wortsman et al., 2022a).

Recent work considers model merging as a way to combine capabilities for models that were fine-tuned on separate tasks from the same base model. Fisher-weighted averaging weights parameters using the empirical Fisher Information, used as a local approximation to the curvature of each model's loss (Matena & Raffel, 2022). Task arithmetic represents the parameter displacement through fine-tuning as a "task vector", computed by taking the difference between the adapted model and the shared pretrained initialization and shows that model capabilities can be composed by adding, subtracting, or scaling such vectors (Ilharco et al., 2023). These methods extend model merging in weight space beyond making fine-tuning more robust on a single task toward multi-task model merging, with the goal of integrating complementary capabilities learned on different tasks.

A central challenge in this setting is interference between task-specific updates, which especially occurs when multiple models are merged. TIES-Merging addresses this issue by trimming updates with small magnitudes to resolve sign conflicts and by averaging only aligned parameter changes (Yadav et al., 2023). DARE, on the other hand, exploits redundancy in fine-tuning deltas by randomly dropping and rescaling parameter differences before merging (Yu et al., 2024a).

Most of this literature studies merging as a post-hoc operation over a fixed set of fine-tuned models. In contrast to this, we consider sequential adaptation to sequences, where each contains data sampled i.i.d. from the targeted distribution but not from the overall data mixture. Each sequence-specific local update must then be integrated into a continuing model. This changes the central question from *how to merge a collection of experts* to *how much of a sequence-specific update should become permanent* while preserving general performance. More recent work applies model merging sequentially in continual-learning settings. We discuss these methods and their relation to our setting in Section 2.

**Linear mode connectivity** Since the loss of a deep neural network is highly non-linear in the model parameters, it is not obvious that linear interpolation between checkpoints should preserve model performance. Early work on mode connectivity showed that optima obtained through different independent training runs on the same data usually are connected through continuous paths in the loss landscape. Garipov et al. found that such minima that share the same architecture can be connected by simple curves in weight space, such as polygonal chains or Bézier curves, along which both training and test loss remain low (Garipov et al., 2018). Draxler et al. observed similar behaviors using a nudged elastic band method to find continuous paths between minima, showing that apparent barriers between solutions can often be removed by following a *non-linear* path through parameter space (Draxler et al., 2018).

These results suggest that neural network solutions may belong to connected low-loss regions rather than isolated basins. Importantly, they do not imply that weights on the straight line between two checkpoints result in low loss. This stronger property, known as *linear mode connectivity*, is more directly relevant for linear model merging, since simple weight averaging evaluates points on the linear interpolation path

$$\theta(\lambda) = (1-\lambda)\theta_a + \lambda\theta_b, \qquad \lambda \in [0,1]. \tag{7}$$

Frankle et al. connect linear mode connectivity to training stability (Frankle et al., 2020). They show that, after an early phase of shared training, independently continued runs from the same checkpoint become stable to SGD noise: even though the data order in the continued runs differ, the final models are connected by a low-loss linear path. This is particularly relevant for pretrained models that form their representation from a very long initial phase of initial training. Thus, fine-tuned checkpoints share a long optimization history and differ only by the additional adaptation update. In such a setting, linear interpolation is therefore more plausible than in the case of independently trained models.

A second reason why linear connectivity may be more likely in modern language models is overparameterization. Entezari et al. (2022) and Ainsworth et al. (2023) suggest that sufficiently wide networks can admit low-barrier linear interpolation paths after their parameters are aligned. In this view, model width and training time are important factors for the emergence of linear connectivity. Although this does not guarantee that arbitrary fine-tuned checkpoints can be merged safely, it suggests that large pretrained models are a promising setting for linear weight-space interpolation.

Together, these observations motivate linear model merging as a promising integration rule for continual fine-tuning of language models. After each non-i.i.d. sequence, local adaptation produces an endpoint $\theta_t^{(L)}$ that shares almost all of its training history with the continuing model $\theta_t$. If the

sequence update remains in a linearly connected region, then partial integration may retain useful local information at very low computational cost.

**Meta-learning connections** The task-vector integration rule considered in this work also appears in optimization-based meta-learning. In particular, Reptile (Nichol et al., 2018) alternates between local adaptation and an outer meta update. Starting from parameters $\theta$, it performs several SGD steps on a sampled task to obtain adapted parameters $\theta'$, and then applies

$$\theta \leftarrow \theta + \lambda(\theta' - \theta) = (1 - \lambda)\theta + \lambda\theta'. \tag{8}$$

This update formulation has the same form as the method we study here: scaling the task vector obtained from local adaptation before integrating it into the continuing model. However, the two settings assign different meanings to the update. Reptile uses it to learn an initialization that can be adapted quickly to a new task, whereas in our continual learning setting it determines how much of the current sequence-specific adaptation should be incorporated permanently.

Reptile is closely related to First-Order MAML (FOMAML) (Finn et al., 2017), which uses the target-loss gradient at the final adapted parameters, whereas Reptile uses the displacement produced by the task-specific adaptation. As detailed in Appendix B.4, these directions are not identical. Nichol et al. (2018) show in a leading-order analysis that the connection between Reptile and FOMAML comes from shared within-task gradient-alignment terms.

The use of displacements collected over multiple sequential local updates also appears in federated learning. `FedAvg` performs several optimization steps on decentralized client data and aggregates the resulting client parameters at a server (McMahan et al., 2017). It addresses distributed aggregation across clients, whereas we apply scaled task vectors sequentially within one continuing non-i.i.d. stream.

# B Analysis

## B.1 Task-vector scaling versus learning-rate scaling

We analyze Task Vector Descent when the $L$ minibatch updates are performed using stochastic gradient descent. At integration step $t$, the minibatches $B_{t,1}, \ldots, B_{t,L}$ contain samples from the same targeted distribution. Starting from $\theta_t$, the trajectory is

$$\theta_t^{(0)} = \theta_t, \tag{9}$$

$$\theta_t^{(i)} = \theta_t^{(i-1)} - \eta \underbrace{\nabla_{\theta_t^{(i-1)}} \mathcal{L}\left(\theta_t^{(i-1)}; B_{t,i}\right)}_{:= g_i}, \qquad i = 1, \ldots, L. \tag{10}$$

The task vector is the accumulated displacement

$$\Delta_t = \theta_t^{(L)} - \theta_t = -\eta \sum_{i=1}^{L} g_i. \tag{11}$$

We approximate $g_i$ to second order around $\theta_t$:

$$g_i = \bar{g}_i + \bar{H}_i \left(\theta_t^{(i-1)} - \theta_t\right) + \mathcal{O}\left(\left\|\theta_t^{(i-1)} - \theta_t\right\|^2\right), \tag{12}$$

where $\bar{g}_i := \nabla_\theta \mathcal{L}(\theta_t; B_{t,i})$ and $\bar{H}_i := \nabla_\theta^2 \mathcal{L}(\theta_t; B_{t,i})$. Assuming fixed $L$ and bounded gradients, $\theta_t^{(i-1)} - \theta_t = \mathcal{O}(\eta)$, and therefore

$$\theta_t^{(i-1)} - \theta_t = -\eta \sum_{j=1}^{i-1} g_j \tag{13}$$

$$= -\eta \sum_{j=1}^{i-1} \bar{g}_j + \mathcal{O}(\eta^2). \tag{14}$$

Substitution into the Taylor expansion gives

$$g_i = \bar{g}_i - \eta \bar{H}_i \sum_{j=1}^{i-1} \bar{g}_j + \mathcal{O}(\eta^2). \tag{15}$$

For any sequence $s$ and parameters $\theta$, we define its accumulated gradient and sequential-update correction as

$$G_s(\theta) \coloneqq \sum_{i=1}^{L} \nabla_\theta \mathcal{L}(\theta; B_{s,i}), \tag{16}$$

$$C_s(\theta) \coloneqq \sum_{i=1}^{L} \nabla_\theta^2 \mathcal{L}(\theta; B_{s,i}) \sum_{j=1}^{i-1} \nabla_\theta \mathcal{L}(\theta; B_{s,j}). \tag{17}$$

The correction $C_s(\theta)$ captures within-sequence path dependence. Updates on earlier minibatches change the parameters $\theta$ thereby changing the gradients computed later. The Hessian–gradient products correct for these changes to first order.

**Task Vector Descent** The task vector obtained at learning rate $\eta$ is

$$\Delta_t = -\eta G_t(\theta_t) + \eta^2 C_t(\theta_t) + \mathcal{O}(\eta^3). \tag{18}$$

After scaling the task vector by $\lambda$, the continuing-model update is

$$\Delta_t^{\mathrm{TVD}} \coloneqq \lambda \Delta_t = \lambda \left( \theta_t^{(L)} - \theta_t \right) \tag{19}$$

$$= -\lambda\eta \underbrace{G_t(\theta_t)}_{\text{accumulated gradient}} + \lambda\eta^2 \underbrace{C_t(\theta_t)}_{\text{sequential-update correction}} + \mathcal{O}(\eta^3). \tag{20}$$

**Learning-rate scaling** For comparison, consider fully integrating the displacement produced by the $L$ minibatch updates using the smaller learning rate $\lambda\eta$. Starting from $\widetilde{\theta}_t^{(0)} = \theta_t$, let

$$\widetilde{\theta}_t^{(i)} = \widetilde{\theta}_t^{(i-1)} - \lambda\eta \nabla_\theta \mathcal{L} \left( \widetilde{\theta}_t^{(i-1)}; B_{t,i} \right). \tag{21}$$

The resulting update is

$$\Delta_t^{\mathrm{LR}} \coloneqq \widetilde{\theta}_t^{(L)} - \theta_t \tag{22}$$

$$= -\lambda\eta \underbrace{G_t(\theta_t)}_{\text{accumulated gradient}} + \lambda^2\eta^2 \underbrace{C_t(\theta_t)}_{\text{sequential-update correction}} + \mathcal{O}(\eta^3). \tag{23}$$

This shows that task-vector scaling is not equivalent to reducing the learning rate. Both methods scale the accumulated gradient by $\lambda$, but task-vector scaling preserves the relative contribution of interactions generated along the original optimization trajectory, whereas learning-rate scaling suppresses them by an *additional factor* of $\lambda$.

### B.2 Interactions across consecutive integration steps

We next distinguish interactions among the $L$ minibatches drawn from one targeted distribution from interactions across two consecutive integration steps. For $s \in \{t, t+1\}$, let $G_s$ and $C_s$ be the quantities defined above and define the accumulated Hessian

$$H_s(\theta) \coloneqq \nabla_\theta G_s(\theta) = \sum_{i=1}^{L} \nabla_\theta^2 \mathcal{L}(\theta; B_{s,i}). \tag{24}$$

For a generic learning rate $\rho$, let $\Delta_s^{(\rho)}(\theta)$ denote the displacement obtained by processing $B_{s,1}, \ldots, B_{s,L}$ from starting parameters $\theta$. The preceding analysis gives

$$\Delta_s^{(\rho)}(\theta) = -\rho G_s(\theta) + \rho^2 C_s(\theta) + \mathcal{O}(\rho^3). \tag{25}$$

To derive both update rules at once, let $\kappa$ denote the coefficient applied to each resulting displacement. Starting from $\theta_t$, two consecutive updates are

$$\theta_{t+1}^{(\kappa,\rho)} = \theta_t + \kappa\Delta_t^{(\rho)}(\theta_t), \tag{26}$$

$$\theta_{t+2}^{(\kappa,\rho)} = \theta_{t+1}^{(\kappa,\rho)} + \kappa\Delta_{t+1}^{(\rho)}\left(\theta_{t+1}^{(\kappa,\rho)}\right). \tag{27}$$

To express both displacements at the common reference point $\theta_t$, we expand the second displacement as

$$\Delta_{t+1}^{(\rho)}\left(\theta_{t+1}^{(\kappa,\rho)}\right) = \Delta_{t+1}^{(\rho)}(\theta_t) + \nabla_\theta\Delta_{t+1}^{(\rho)}(\theta_t)\left(\theta_{t+1}^{(\kappa,\rho)} - \theta_t\right) + \mathcal{O}(\rho^3). \tag{28}$$

The remainder is $\mathcal{O}(\rho^3)$ because $\theta_{t+1}^{(\kappa,\rho)} - \theta_t = \mathcal{O}(\rho)$ and $D^2\Delta_{t+1}^{(\rho)} = \mathcal{O}(\rho)$. From the displacement expansion,

$$\nabla_\theta\Delta_{t+1}^{(\rho)}(\theta_t) = -\rho H_{t+1}(\theta_t) + \mathcal{O}(\rho^2), \tag{29}$$

$$\theta_{t+1}^{(\kappa,\rho)} - \theta_t = -\kappa\rho G_t(\theta_t) + \mathcal{O}(\rho^2). \tag{30}$$

Therefore,

$$\Delta_{t+1}^{(\rho)}\left(\theta_{t+1}^{(\kappa,\rho)}\right) = \Delta_{t+1}^{(\rho)}(\theta_t) + \kappa\rho^2 H_{t+1}(\theta_t)G_t(\theta_t) + \mathcal{O}(\rho^3). \tag{31}$$

Training on sequence $t$ displaces the parameters from which sequence $t+1$ begins. The product $H_{t+1}(\theta_t)G_t(\theta_t)$ captures this dependence. Substitution into the two-update recursion gives

$$\begin{aligned}
\theta_{t+2}^{(\kappa,\rho)} - \theta_t = &-\kappa\rho\,(G_t + G_{t+1}) && (32)\\
&+\kappa\rho^2\,(C_t + C_{t+1}) && (33)\\
&+\kappa^2\rho^2 H_{t+1}G_t && (34)\\
&+\mathcal{O}(\rho^3), && (35)
\end{aligned}$$

where all quantities on the right-hand side are evaluated at $\theta_t$.

**Task Vector Descent** corresponds to $(\kappa, \rho) = (\lambda, \eta)$, yielding

$$\begin{aligned}
\theta_{t+2} - \theta_t = &-\lambda\eta\,(G_t + G_{t+1}) && (36)\\
&+\boldsymbol{\lambda}\eta^2\,(C_t + C_{t+1}) && (37)\\
&+\lambda^2\eta^2 H_{t+1}G_t && (38)\\
&+\mathcal{O}(\eta^3). && (39)
\end{aligned}$$

**Learning-rate scaling** corresponds to $(\kappa, \rho) = (1, \lambda\eta)$, yielding

$$\begin{aligned}
\widetilde{\theta}_{t+2} - \theta_t = &-\lambda\eta\,(G_t + G_{t+1}) && (40)\\
&+\boldsymbol{\lambda^2}\eta^2\,(C_t + C_{t+1}) && (41)\\
&+\lambda^2\eta^2 H_{t+1}G_t && (42)\\
&+\mathcal{O}(\eta^3). && (43)
\end{aligned}$$

The two methods have the same first-order update ($G_t+G_{t+1}$) and scale the cross-sequence interaction ($H_{t+1}G_t$) by $\lambda^2$. They differ only in the sequence-dependent correction ($C_t + C_{t+1}$), highlighted in **blue**.

Task Vector Descent computes the local trajectory at the original learning rate before scaling the resulting displacement, so the correction term is multiplied by $\lambda$. In contrast, learning-rate scaling scales every minibatch step, resulting in a factor of $\lambda^2$. Thus, Task Vector Descent preserves more within-sequence optimization structure without strengthening cross-sequence coupling (to second order).

### B.3 Optimal scaling under a local quadratic model

The preceding analysis distinguishes task-vector scaling from learning-rate scaling. We now ask when partial integration can improve the average objective over full integration. Related curvature-based optimal coefficients have been derived for continual model merging (Qiu et al., 2025; Zhang et al., 2019). Here, we apply the local quadratic perspective to the average population loss along a sequence task vector. Let

$$F(\theta) \coloneqq \mathbb{E}_k \mathbb{E}_{z\sim P_k}\left[\mathcal{L}(\theta; z)\right] \tag{44}$$

denote the average population loss from Eq. (1). Along the task vector $\Delta_t$, a second-order expansion around $\theta_t$ gives

$$F(\theta_t + \lambda\Delta_t) = F(\theta_t) + \lambda \nabla F(\theta_t)^\top \Delta_t + \frac{\lambda^2}{2}\Delta_t^\top \nabla^2 F(\theta_t)\Delta_t + o\left(\|\Delta_t\|^2\right), \tag{45}$$

The linear term is the first-order change in average loss along the task vector, whereas the quadratic term captures the curvature in that direction. If $\Delta_t^\top \nabla^2 F(\theta_t)\Delta_t > 0$, differentiating Eq. (45) with respect to $\lambda$ gives the constrained quadratic-model optimum

$$\lambda_t^\star = \mathrm{clip}_{[0,1]}\left(-\frac{\nabla F(\theta_t)^\top \Delta_t}{\Delta_t^\top \nabla^2 F(\theta_t)\Delta_t}\right). \tag{46}$$

In the case of $\nabla F(\theta_t)^\top \Delta_t \geq 0$, moving along the task vector does not decrease the average loss even to first order, and the quadratic model prefers $\lambda = 0$. If this directional derivative is negative and $0 < \lambda_t^* < 1$, moving along task vector initially improves the average objective but full integration passes the minimum along that direction, making partial integration optimal. If the ratio is at least one, the quadratic-model minimum lies at or beyond the locally adapted endpoint, and full integration is optimal.

Under the simplifying assumption that longer same-distribution sequences primarily increase the task-vector magnitude while preserving its direction and the local geometry, the linear term grows with $\|\Delta_t\|$, whereas the curvature term grows with $\|\Delta_t\|^2$. The preferred coefficient, therefore, decreases approximately inversely with task-vector magnitude, providing a local explanation for why partial integration becomes more useful for longer sequences. Equation (46) is an oracle characterization rather than an online selection rule because the average-loss gradient and curvature are unavailable from the targeted distribution alone.

### B.4 Relation between Reptile and FOMAML

Reptile's connection to FOMAML becomes clearer by expanding the local optimization. Let

$$\theta^{(0)} = \theta, \tag{47}$$

$$\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla_\theta \mathcal{L}_{\tau,i}\left(\theta^{(i)}\right), \qquad i = 0, \ldots, k-1, \tag{48}$$

where $\mathcal{L}_{\tau,i}$ denotes the loss on the minibatch used in local step $i$. Let $\mathcal{L}_\tau^*$ denote the target loss evaluated after adaptation. MAML minimizes

$$\mathcal{J}_{\mathrm{MAML}}(\theta^{(0)}) = \mathbb{E}_\tau\left[\mathcal{L}_\tau^*\left(\theta^{(k)}\right)\right], \tag{49}$$

whose gradient is

$$\nabla_\theta^{(0)} \mathcal{J}_{\mathrm{MAML}}(\theta^{(0)}) = \mathbb{E}_\tau\left[\left(\frac{\partial \theta^{(k)}}{\partial \theta^{(0)}}\right)^\top \frac{\partial}{\partial \theta^{(k)}}\mathcal{L}_\tau^*\left(\theta^{(k)}\right)\right]. \tag{50}$$

FOMAML approximates the adaptation Jacobian by the identity, yielding the meta-gradient

$$g_{\mathrm{FOMAML}} = \mathbb{E}_\tau\left[\nabla_{\theta^{(k)}}\mathcal{L}_\tau^*\left(\theta^{(k)}\right)\right]. \tag{51}$$

Thus, FOMAML computes a gradient at the final adapted parameters using a separate target batch. Reptile instead uses the expected local parameter displacement,

$$g_{\mathrm{Reptile}} \coloneqq -\frac{1}{\alpha}\mathbb{E}_\tau\left[\theta^{(k)} - \theta^{(0)}\right] \tag{52}$$

$$= \mathbb{E}_\tau\left[\sum_{i=0}^{k-1} \nabla_\theta \mathcal{L}_{\tau,i}\left(\theta^{(i)}\right)\right]. \tag{53}$$

To make the corresponding first-order approximation explicit, define the cumulative loss along the local trajectory as

$$\mathcal{J}_{\text{trajectory}}(\theta) \coloneqq \mathbb{E}_\tau \left[ \sum_{i=0}^{k-1} \mathcal{L}_{\tau,i} \left( \theta^{(i)} \right) \right]. \tag{54}$$

Its exact gradient reintroduces the adaptation Jacobian at every local step:

$$\nabla_\theta \mathcal{J}_{\text{trajectory}}(\theta) = \mathbb{E}_\tau \left[ \sum_{i=0}^{k-1} \left( \frac{\partial \theta^{(i)}}{\partial \theta} \right)^\top \frac{\partial}{\partial \theta^{(i)}} \mathcal{L}_{\tau,i} \left( \theta^{(i)} \right) \right]. \tag{55}$$

Approximating each adaptation Jacobian by the identity gives

$$\nabla_\theta \mathcal{J}_{\text{trajectory}}(\theta) \approx \mathbb{E}_\tau \left[ \sum_{i=0}^{k-1} \nabla_{\theta^{(i)}} \mathcal{L}_{\tau,i} \left( \theta^{(i)} \right) \right] = g_{\text{Reptile}}. \tag{56}$$

This has the same chain-rule form as the MAML gradient in Eq. (50), but replaces the terminal target loss with the cumulative loss along the local trajectory. Reptile can therefore be interpreted as a first-order gradient of this trajectory objective (Huisman et al., 2024), which differs from the terminal target-loss objective used by FOMAML. Their connection also appears in leading-order expansions: both directions contain terms that encourage gradients from different minibatches of the same task to align, although these terms receive different weights (Nichol et al., 2018).

# C Complete method specification

This section provides the complete specification of the training procedure introduced in Section 3. We first describe the sequence-level procedure shared across settings, followed by the AdamW updates and optimizer-state integration. We then specify the local adaptation objectives and linear parameter integration.

## C.1 Sequence-level training procedure

Every run trains on a stream of $T$ sequences. In the fixed-distribution settings, sequence $t$ contains $L$ consecutive minibatches $B_{t,1}, \ldots, B_{t,L}$ from the targeted distribution $P_{k_t}$. A minibatch is the setting-specific atomic training unit: a packed-token minibatch for language modelling, a minibatch of prompt–response pairs for supervised fine-tuning, or a prompt minibatch with task-specific answer verifiers for GRPO. For persona learning, one prompt interaction is the atomic unit. Thus, $L$ always controls how long the model processes one targeted distribution or collection context before an integration step, although the amount of computation per unit differs between settings.

We denote the continuing checkpoint entering sequence $t$ by $(\theta_t, \omega_t)$, where $\theta_t$ contains the model parameters and $\omega_t$ the optimizer state. Starting from $(\theta_t^{(0)}, \omega_t^{(0)}) = (\theta_t, \omega_t)$, the setting-specific optimizer processes the $L$ training units and produces the locally adapted endpoint $\theta_t^{(L)}$ and optimizer state $\omega_t^{(L)}$. We then integrate both into the next continuing checkpoint using the coefficient $\lambda$. For each sequence, we evaluate the continuing model before local adaptation, the locally adapted endpoint, and the continuing model after integration, as shown in Fig. 2. Algorithm 1 summarizes the complete procedure.

## C.2 AdamW local updates

The update in Eq. (2) illustrates local adaptation using SGD without momentum. We use AdamW (Loshchilov & Hutter, 2019) in our experiments. At the start of sequence $t$, we inherit the full continuing checkpoint,

$$\left( \theta_t^{(0)}, m_t^{(0)}, v_t^{(0)}, c_t^{(0)} \right) = (\theta_t, m_t, v_t, c_t). \tag{57}$$

Here, $m$ and $v$ are the first and second gradient moments and $c$ is the optimizer step count. For each executed optimizer step $i$, let $g_t^{(i)}$ be the gradient of the setting-specific loss on the effective

**Algorithm 1** Continual training with scaled task-vector updates

**Require:** Initial parameters $\theta_0$, optimizer state $\omega_0$, minibatches $B_{t,i}$, number $L$ of consecutive minibatches, coefficient $\lambda$

**for** $t = 0, \dots, T-1$ **do**
  $(\theta_t^{(0)}, \omega_t^{(0)}) \leftarrow (\theta_t, \omega_t)$
  Record evaluation of $\theta_t$
  **for** $i = 1, \dots, L$ **do**
    $(\theta_t^{(i)}, \omega_t^{(i)}) \leftarrow \mathrm{OptimizerStep}(\theta_t^{(i-1)}, \omega_t^{(i-1)}; B_{t,i})$
  **end for**
  $\Delta_t \leftarrow \theta_t^{(L)} - \theta_t$
  Record evaluation of $\theta_t^{(L)}$
  $\theta_{t+1} \leftarrow \theta_t + \lambda \Delta_t$
  $\omega_{t+1} \leftarrow \mathrm{MergeAdamW}(\omega_t, \omega_t^{(L)}; \lambda)$
  Record evaluation of $\theta_{t+1}$
**end for**
**return** $(\theta_T, \omega_T)$

minibatch, after gradient accumulation and gradient-norm clipping where applicable. For a parameter group with learning rate $\eta_t^{(i)}$, moment coefficients $\beta_1, \beta_2 \in [0, 1)$, weight-decay coefficient $\rho \geq 0$, and numerical stabilizer $\epsilon > 0$, the updates are

$$c_t^{(i)} = c_t^{(i-1)} + 1, \tag{58}$$

$$m_t^{(i)} = \beta_1 m_t^{(i-1)} + (1 - \beta_1) g_t^{(i)}, \tag{59}$$

$$v_t^{(i)} = \beta_2 v_t^{(i-1)} + (1 - \beta_2) \left( g_t^{(i)} \right)^2 \tag{60}$$

$$\widehat{m}_t^{(i)} = \frac{m_t^{(i)}}{1 - \beta_1^{c_t^{(i)}}}, \qquad \widehat{v}_t^{(i)} = \frac{v_t^{(i)}}{1 - \beta_2^{c_t^{(i)}}}, \tag{61}$$

$$\theta_t^{(i)} = \left( 1 - \eta_t^{(i)} \rho \right) \theta_t^{(i-1)} - \eta_t^{(i)} \frac{\widehat{m}_t^{(i)}}{\sqrt{\widehat{v}_t^{(i)}} + \epsilon}. \tag{62}$$

Squares and square roots are applied elementwise. The bias-correction factors use the inherited count $c_t^{(i)}$, which is not reset at sequence boundaries and can be fractional after the optimizer-state interpolation. After local adaptation, the resulting parameters and optimizer state are integrated as described in Appendix C.3. The learning rates, moment coefficients, weight decay, and clipping settings are specified in Appendix D.

### C.3 Optimizer-state integration

Because the model parameters are linearly interpolated between the continuing-model parameters $\theta_t$ and locally adapted endpoint $\theta_t^{(L)}$, we apply the same interpolation to the optimizer state. In AdamW, the optimizer state in the continuing checkpoint $\omega_t = (m_t, v_t, c_t)$ is represented by the first moment, second moment, and step count. Let $\omega_t^{(L)} = (m_t^{(L)}, v_t^{(L)}, c_t^{(L)})$ denote the corresponding state after processing the current sequence. We obtain the continuing optimizer state by linearly interpolating the first moment and step count and interpolating the second moment in root-mean-square space:

$$m_{t+1} = (1 - \lambda) m_t + \lambda m_t^{(L)}, \tag{63}$$

$$c_{t+1} = (1 - \lambda) c_t + \lambda c_t^{(L)}, \tag{64}$$

$$v_{t+1} = \left( (1 - \lambda) \sqrt{v_t} + \lambda \sqrt{v_t^{(L)}} \right)^2. \tag{65}$$

We use the same coefficient $\lambda$ to integrate model parameters and the optimizer state to ensure consistency between the partially integrated parameters and the AdamW state. All other optimizer

hyperparameters and the learning-rate schedule remain constant and require no special handling; non-floating model-state entries remain unchanged. At $\lambda = 1$, these rules return the parameters and AdamW state produced by the local updates exactly, reproducing the standard continual-training baseline.

### C.4 Local adaptation objectives

We use four local adaptation objectives: next-token language modelling for continual pretraining and pretraining from random initialization, supervised fine-tuning to teach the model chain-of-thought reasoning, SDPO to adapt to a user's persona, and GRPO to learn from verifiable rewards. The corresponding data and hyperparameters are specified in Appendix D.

**Next-token language modelling** In language modelling, a minibatch $B$ contains packed token sequences $z = (z_1, \ldots, z_{|z|})$. Training is performed by minimizing the causal negative log likelihood

$$\mathcal{L}_{\mathrm{LM}}(\theta; B) = -\frac{1}{n_B^{\mathrm{tok}}} \sum_{z \in B} \sum_{j=2}^{|z|} \log \pi_\theta(z_j \mid z_{<j}), \tag{66}$$

$$n_B^{\mathrm{tok}} = \sum_{z \in B} (|z| - 1).$$

The loss is normalized by the number of target tokens. Each minibatch corresponds to one AdamW update; therefore, $L$ equals both the number of minibatches and the number of gradient steps per sequence in the pretraining settings.

**Reasoning with supervised fine-tuning** For reasoning SFT, each example consists of a prompt $x$ and an expert response $y = (y_1, \ldots, y_{|y|})$, often sourced from domain experts or a stronger teacher model. The response contains a reasoning trace followed by a final answer. Training is performed by minimizing the cross-entropy on the completion tokens $y$ only:

$$\mathcal{L}_{\mathrm{SFT}}(\theta; B) = -\frac{1}{M_B} \sum_{(x,y) \in B} \sum_{j=1}^{|y|} \log \pi_\theta(y_j \mid x, y_{<j}), \tag{67}$$

$$M_B = \sum_{(x,y) \in B} |y|.$$

Here, the prompt tokens $x$ serve as the conditioning context but are masked from the loss computation. Unlike the on-policy objectives below, SFT trains on fixed off-policy traces rather than on responses generated by the model.

**Persona learning with SDPO** In this setting, the goal is to adapt a policy to a user's preferences. The preferences are only implicitly contained in the user's textual feedback. First, the policy generates a response $y$ to a user prompt $x$. A user simulator conditioned on the persona targeted in the current sequence then returns either `ACCEPT` or textual feedback $f$. If the response is accepted, no gradient update is performed. If the response is rejected, the previous response is scored under the feedback-conditioned distribution $\pi_\theta(\cdot \mid x, f, y_{<j})$. The loss is the token-averaged reverse KL divergence

$$\mathcal{L}_{\mathrm{SDPO}}(\theta; x, y, f) = \frac{1}{|y|} \sum_{j=1}^{|y|} D_{\mathrm{KL}}(\widetilde{\pi}_\theta(\cdot \mid x, y_{<j}) \,\|\, \mathrm{sg}[\widetilde{\pi}_\theta(\cdot \mid x, f, y_{<j})]) \,. \tag{68}$$

Here, sg is the stop-gradient operation and $\widetilde{\pi}$ denotes the compressed token distribution that retains the union of the 20 most likely tokens under the student and feedback-conditioned distributions at each position and aggregates the remaining probability mass into one tail bin. Intuitively, this objective compresses the simulator's feedback into the feedback-free student policy while keeping the student-generated response fixed as the states on which both distributions are compared.

**Reasoning with GRPO** For GRPO (Shao et al., 2024), the policy first generates a group of $G$ answers $y^{(j)}$ for a given question $x$. Each answer is then verified and assigned the reward

$$r_j = \mathbf{1}[\mathrm{correct}(y^{(j)}, y^*)] + \beta_{\mathrm{fmt}} \mathbf{1}[\mathrm{valid_format}(y^{(j)})]. \tag{69}$$

Correctness and validity of the answer format are determined by the task-specific verifier and format requirements. The rewards are normalized relative to the other responses in the group:

$$\widehat{A}_j = \frac{r_j - \frac{1}{G}\sum_{h=1}^{G} r_h}{\mathrm{std}(r_1, \ldots, r_G) + \delta}, \tag{70}$$

where $\delta > 0$ ensures numerical stability. Let

$$\rho_{j,q}(\theta) = \frac{\pi_\theta(y_q^{(j)} \mid x, y_{<q}^{(j)})}{\pi_{\theta_{\mathrm{old}}}(y_q^{(j)} \mid x, y_{<q}^{(j)})} \tag{71}$$

denote the importance ratio between the updated and rollout policies. We use a DAPO-style clipped policy loss with active-token normalization (Yu et al., 2025):

$$\mathcal{L}_{\mathrm{GRPO}}(\theta; B) = -\frac{1}{N_B^{\mathrm{tok}}} \sum_{x \in B} \sum_{j=1}^{G} \sum_{q=1}^{|y^{(j)}|} \min\Big(\rho_{j,q}(\theta)\widehat{A}_j, \mathrm{clip}(\rho_{j,q}(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}})\,\widehat{A}_j\Big), \tag{72}$$

where $N_B^{\mathrm{tok}}$ is the number of active completion tokens. We use eight on-policy rollouts per prompt, $\epsilon_{\mathrm{low}} = 0.2$, and $\epsilon_{\mathrm{high}} = 0.28$. We use neither a reference-model KL penalty nor a reasoning-length reward.

### C.5 Linear parameter integration

Each of the four adaptation procedures produces the locally adapted endpoint and optimizer state $(\theta_t^{(L)}, \omega_t^{(L)})$. The local update results in a parameter displacement between the locally adapted endpoint and the continuing-model parameters,

$$\Delta_t = \theta_t^{(L)} - \theta_t. \tag{73}$$

We integrate the task vector by moving the parameters $\theta_t$ in the direction of this displacement:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \lambda \Delta_t \\ &= (1-\lambda)\theta_t + \lambda \theta_t^{(L)}, \qquad \lambda \in [0, 1]. \end{aligned} \tag{74}$$

Setting $\lambda = 0$ discards the local update, whereas full integration ($\lambda = 1$) carries the locally adapted endpoint forward without attenuation. As discussed in Appendix B.4, this update is equivalent to a Reptile-style meta-update. We use it here as a general integration mechanism rather than as a meta-learning algorithm.

Linear parameter integration requires only one additional hyperparameter, the coefficient $\lambda$. It also requires the continuing checkpoint to be retained while processing the current sequence.

## D Complete experimental configurations

This section specifies the data, models, hyperparameters, stream construction, and evaluation protocol used to instantiate the methods from Section 3. We study task-vector scaling in six settings: continual pretraining, persona learning with SDPO, reasoning SFT, reasoning GRPO, pretraining from random initialization, and book-level continual pretraining. Table 1 summarizes the experimental conditions.

### D.1 Common protocol

The experiments share a common training and evaluation setup. Except for PG-19, the selected training examples are fixed across seeds. The seeds instead determine the sequence order, training randomness, and, where applicable, stochastic generation. Each sequence contains training units from a single targeted distribution or collection context. Within a run, we use a static integration coefficient $\lambda$ for all sequences.

After local adaptation, we integrate the model parameters according to Eq. (3) and the AdamW optimizer state according to Eqs. (63) to (65). We evaluate the model at three points around each sequence: before local adaptation, after local adaptation but before integration, and after integration. These evaluations are purely observational and do not affect training or the choice of $\lambda$.

Table 1: Linear-integration experiments. In the fixed-distribution settings, $L$ counts consecutive minibatches from one targeted distribution. In the on-policy extensions, it counts consecutive setting-specific collection units from one active context. For PG-19, sequence lengths vary with book length.

| Setting | Model | $K$ | $L$ | $\lambda$ | # Seeds | Primary score |
|---|---|---|---|---|---|---|
| Continual PT | GPT-2 | 10 | $1, 32, 64, 128, 256, 512$ | $.4, .6, .8, 1$ | 3 | BPT gain |
| Persona SDPO | Qwen3-4B | 6 | $4, 8, 16$ | $0, .01, .05, .1, .2, .4, .6, .8, 1$ | 3 | Macro win rate |
| Reasoning SFT | Qwen2.5-3B | 4 | $32, 64, 128, 256$ | $0, .2, .4, .6, .8, 1$ | 5 | Best4 |
| Reasoning GRPO | Qwen2.5-3B | 4 | $256, 1024$ | $0, .2, .4, .6, .8, 1$ | 3 | Best4 |
| Scratch PT | GPT-2 | 10 | $1, 64, 256, 1024, 2048$ | $.2, .4, .6, .8, 1$ | 3 | BPT gain |
| Book-level PT | SmolLM-360M | 200 | 1–158 | $.4, .6, .8, 1$ | 3 | BPT gain |

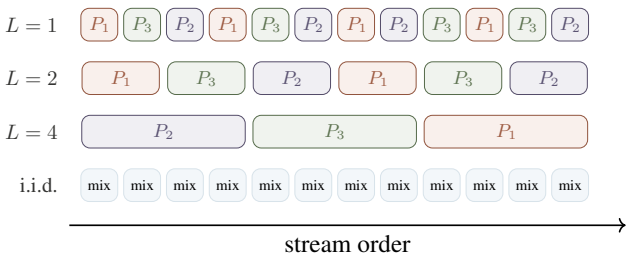


Figure 9: Controlled streams. Each box contains $L$ batches from one distribution; i.i.d. batches draw from the complete mixture. Larger $L$ increases temporal clustering.

**Aggregation and uncertainty** At each evaluation checkpoint, we first average uniformly over the evaluated examples within each distribution and then compute an unweighted macro-average across distributions. Unless stated otherwise, the run-level estimate is the mean over all evaluated sequences. This aggregation prevents distributions with more evaluation examples and the final sequence from disproportionately influencing the reported result. We aggregate independent runs using the arithmetic mean and report the sample standard error of the mean (SEM),

$$\mathrm{SEM} = \frac{s}{\sqrt{n}}, \tag{75}$$

where $s$ is the sample standard deviation of the $n$ run-level estimates.

**Evaluation metrics** For language modelling, we report bits per token (BPT). Given the mean token-level negative log likelihood $\mathcal{L}_{\mathrm{token}}$ in nats, BPT is defined as

$$\mathrm{BPT} = \frac{\mathcal{L}_{\mathrm{token}}}{\ln 2}. \tag{76}$$

Because lower BPT indicates better performance, we use $\mathrm{score}(\theta; k) = -\mathrm{BPT}_k(\theta)$, consistent with the higher-is-better convention in Section 3. The resulting gain is the BPT reduction relative to the initial checkpoint $\theta_0$:

$$\mathrm{gain}(\theta; k) = \mathrm{BPT}_k(\theta_0) - \mathrm{BPT}_k(\theta). \tag{77}$$

Thus, positive values indicate an improvement over the initial checkpoint. We use the same token-normalized metric for continual pretraining and pretraining from random initialization.

For the generative reasoning tasks, we sample four independent responses per prompt. Best@4 measures whether at least one of the four responses is correct, whereas Mean@4 is the mean verifier correctness across the four responses. We use Best@4 as the metric for SFT and GRPO.

**Frameworks** We use Transformers and TRL for model training and vLLM for model inference and evaluation (Wolf et al., 2020; von Werra et al., 2020; Kwon et al., 2023).

**Existing assets** Table 2 lists the models, datasets, and principal software used in our experiments and their published licenses or terms.

Table 2: Existing assets and their published licenses or terms. Asset names link to the source documentation; citations credit the original work.

| **Asset / checkpoint** | **License / terms** |
|---|---|
| *Models and services* | |
| GPT-2 (124M) (Radford et al., 2019) | Modified MIT |
| Qwen3-4B, Qwen3-8B (Yang et al., 2025) | Apache 2.0 |
| Qwen2.5-1.5B, Qwen2.5-7B (Yang et al., 2024) | Apache 2.0 |
| Qwen2.5-3B, Qwen2.5-3B-Instruct (Yang et al., 2024) | Qwen Research License |
| Qwen2.5-Coder-1.5B, -7B (Hui et al., 2024) | Apache 2.0 |
| Qwen2.5-Coder-3B (Hui et al., 2024) | Qwen Research License |
| SmolLM-360M (Ben Allal et al., 2024) | Apache 2.0 |
| SmolLM2-1.7B-Instruct (Ben Allal et al., 2025) | Apache 2.0 |
| Claude Sonnet 4.6 (Anthropic, 2026) | Proprietary service terms[†] |
| *Datasets and task generators* | |
| Pile Uncopyrighted (ten domains) (Gao et al., 2020) | MIT (collection)[‡] |
| PG-19 (Rae et al., 2020) | Apache 2.0 (collection)[‡] |
| HelpSteer (Wang et al., 2024b) | CC BY 4.0 |
| Reasoning Gym (four tasks) (Stojanovski et al., 2025) | Apache 2.0 |
| MetaMathQA (Yu et al., 2024b) | MIT (dataset card)[‡] |
| GSM8K (Cobbe et al., 2021) | MIT |
| MBPP (Austin et al., 2021) | CC BY 4.0 |
| MMLU (Hendrycks et al., 2021) | MIT |
| ARC-Challenge (Clark et al., 2018) | CC BY-SA 4.0 |
| HellaSwag (Zellers et al., 2019) | MIT |
| WinoGrande (Sakaguchi et al., 2020) | Apache 2.0 |
| *Principal software* | |
| Transformers (Wolf et al., 2020), TRL (von Werra et al., 2020), vLLM (Kwon et al., 2023) | Apache 2.0 (each) |

The Qwen Research License permits non-commercial research and evaluation; commercial use requires a separate license. [†]Claude is accessed as a service; the applicable service agreement governs its use and generated outputs. [‡]A collection-level license does not replace the terms of underlying sources. The Pile datasheet declares MIT for the original collection; individual documents retain their underlying rights. MetaMathQA derives examples from GSM8K and MATH.

### D.2 CONTINUAL PRETRAINING

In the continual-pretraining setting, we adapt a pretrained language model to a non-i.i.d. stream of text domains and study whether task-vector scaling preserves performance across domains while incorporating each local update.

**Model** We use a pretrained 124M-parameter GPT-2 checkpoint (Radford et al., 2019), originally pretrained by OpenAI on WebText. We perform training in FP32.

**Data and distributions** We select ten domains from the Pile Uncopyrighted dataset (Gao et al., 2020): *PubMed Central, ArXiv, GitHub, FreeLaw, StackExchange, USPTO Backgrounds, Gutenberg (PG-19), English Wikipedia, Hacker News, and NIH ExPorter*.

We first split each domain into disjoint training and evaluation sets at the document level. We then tokenize the documents using the GPT-2 tokenizer and separate adjacent documents with an end-of-sequence token. The resulting token sequences are packed into blocks of 512 tokens.

For each domain, we use 4,096 blocks for training. With a minibatch size of eight blocks, this yields 512 training minibatches per domain and 5,120 optimizer updates across all ten domains. For evaluation, we use 16 minibatches of eight held-out blocks per domain, corresponding to 128 evaluation blocks per domain.

**Streams** From the selected training blocks, we construct streams with varying degrees of temporal locality controlled by $L$. Each training minibatch contains eight blocks from a single domain, and each sequence consists of $L$ consecutive minibatches from that domain. For each domain, the blocks are shuffled once during preprocessing and grouped into 512 fixed minibatches. For a given $L$, we partition this minibatch sequence into consecutive groups of $L$ and randomize the order of the resulting sequences for each experimental seed, while preserving the minibatch order within each sequence. We consider

$$L \in \{1, 32, 64, 128, 256, 512\}. \tag{78}$$

As an i.i.d. reference, we globally shuffle the same 40,960 training blocks and form mixed-domain minibatches of size eight. This reference uses the same initialization, data, optimizer settings, and number of updates, but trains continuously without parameter or optimizer-state integration. It differs from the $L = 1$, $\lambda = 1$ condition, in which every minibatch remains domain-specific and an integration step is performed after each update.

**Local training and integration** We optimize the next-token language-modelling objective in Eq. (66) using AdamW with a learning rate of $4 \times 10^{-5}$ and a weight decay of $0.01$. We select the learning rate through a grid search on the i.i.d. reference and use the selected value for all non-i.i.d. runs. We apply gradient-norm clipping at $1$ and use a constant learning-rate schedule.

After every sequence, we integrate both the model parameters and the optimizer state. We use the same static coefficient throughout a run and consider

$$\lambda \in \{0.4, 0.6, 0.8, 1\}. \tag{79}$$

**Evaluation and seeds** At each sequence boundary, we evaluate all three checkpoints shown in Fig. 2 on all ten domains using 16 held-out minibatches per domain. We report the macro-average BPT gain across domains and average the run-level result over all evaluated sequences. We run both the non-i.i.d. experiments and the i.i.d. reference with seeds $0$, $1$, and $2$.

### D.3 PERSONA LEARNING WITH SDPO

In the persona-learning setting, we adapt a language model to user-specific style preferences conveyed through textual feedback. Each persona defines one collection context, and SDPO performs local adaptation from interactions with a persona-conditioned user simulator. We study whether task-vector scaling can incorporate the learned preference while retaining performance on the other personas.

**Models** We use Qwen3-4B as the model undergoing continual adaptation (Yang et al., 2025). A single Qwen3-8B model is used both as the persona-conditioned user simulator during training and as the pairwise judge during evaluation. The roles are defined through distinct system prompts. All model computations use BF16.

**Data and personas** Following Kleine Buening et al. (2026), we construct the training data from prompts in the HelpSteer training split (Wang et al., 2024b). We randomly select 32 prompts and reuse the same set for every persona. We define six personas, each corresponding to one style preference:

- *No preface:* You dislike unnecessary opening prefaces such as "Certainly," "Sure," "Absolutely," or "Here's ...". You prefer responses to start directly with the answer.
- *Brief preface:* You prefer responses to begin with a brief friendly preface such as "Sure," "Certainly," "Of course," or "Here's ..." before the main answer.
- *No opening praise:* You dislike opening praise or positive acknowledgements. Start directly with the answer instead of saying things like "Great question" or "Nice idea."
- *Opening praise:* You prefer responses to begin with a brief positive acknowledgement or praise before answering, such as "Great question" or "Nice idea."
- *No emojis:* You prefer responses without emojis.
- *Short, precise sentences:* You prefer short, precise sentences. Avoid long multi-clause sentences and split ideas into separate sentences when possible.

This yields $6 \times 32 = 192$ interactions in total.

**Streams and i.i.d. reference** Each prompt interaction constitutes one training unit. The stream contains each of the 32 selected prompts once for each of the six personas. Each sequence contains $L$ interactions associated with a single persona, and we randomize the sequence order for each experimental seed. We consider

$$L \in \{4, 8, 16\}. \tag{80}$$

As an i.i.d. reference, we randomly shuffle the 192 prompt–persona pairs and use one-interaction sequences with $L = 1$ and $\lambda = 1$.

**Local training and integration** We perform local adaptation using the SDPO objective in Eq. (68). For each interaction, the student generates a response greedily with a maximum of 256 new tokens. If the simulator returns `ACCEPT`, we perform no parameter update. Otherwise, it returns a short corrective user message, which we use as feedback for three SDPO updates with the feedback-conditioned distribution as the self-distillation target.

We optimize the student using AdamW with a learning rate of $5 \times 10^{-6}$, weight decay $0.01$, a minibatch size of one, and gradient-norm clipping at $1$. The maximum context length is 4,096 tokens. Both the simulator and judge use greedy decoding. After every sequence, we integrate both the model parameters and optimizer state and consider

$$\lambda \in \{0, 0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1\}. \tag{81}$$

**Evaluation and seeds** We evaluate on 128 held-out prompts from the HelpSteer validation split. For every prompt and persona, we generate responses from the current checkpoint and the initial Qwen3-4B checkpoint. The evaluated models receive no system prompt; each validation prompt is provided as a user message. The judge receives the corresponding persona description in its system message.

For each prompt, the judge compares the current and initial model responses in both presentation orders. We record a win or loss only when the two judgments consistently favor the same response after accounting for presentation order. Explicit ties and disagreements between the two judgments are treated as ties.

For persona $k$, let $W_k$, $L_k$, and $T_k$ denote the numbers of wins, losses, and ties. We compute the conditional win rate as $100W_k/(W_k + L_k)$, excluding ties from the denominator, and macro-average these rates across personas. This metric measures preference for the current model conditional on a decisive comparison. If $W_k + L_k = 0$, the persona score is undefined and excluded from the macro-average; this case did not occur in the evaluated runs. We run both the non-i.i.d. experiments and the i.i.d. reference with seeds $0$, $1$, and $2$.

### D.4 SHARED REASONING SUITE

The SFT and GRPO experiments use the same model, reasoning tasks, and evaluation protocol. We describe these shared components before specifying the training procedure and hyperparameters for each adaptation objective.

**Model** We use Qwen2.5-3B-Instruct (Yang et al., 2024) and perform all model computations in BF16.

**Data and tasks** The experiments use four procedurally generated tasks from Reasoning Gym (Stojanovski et al., 2025):

- **Countdown:** combine exactly three input numbers using arithmetic operations to obtain a specified target value.
- **Mini-Sudoku:** complete a Sudoku puzzle containing exactly seven empty cells.
- **N-Queens:** complete a $4 \times 4$ board from which one queen has been removed.
- **String Manipulation:** repeatedly transform an initial string of length ten according to three ordered rules until no rule applies or a state repeats.

Each task provides a task-specific verifier that evaluates the correctness of the extracted answer. A response is counted as correct only if its final answer appears within complete `<answer>...</answer>` tags. The acceptable representation inside these tags is determined by the task-specific verifier.

**Evaluation** We evaluate on 100 held-out prompts per task and sample four responses per prompt. We generate responses with a temperature of $0.6$, top-$p = 0.95$, top-$k = 20$, and a maximum response length of 2,048 new tokens. We report Best@4.

### D.5 REASONING WITH SUPERVISED FINE-TUNING

**Data** Using data seed 42, we select 256 training examples for each of the four reasoning tasks, giving 1,024 examples in total. For every prompt, we use Claude Sonnet 4.6 (Anthropic, 2026) to generate a fixed reasoning trace followed by a final answer in `<answer>...</answer>` tags. All 1,024 generated traces pass the corresponding task-specific correctness verifier and contain a valid answer tag.

**Streams and i.i.d. reference** Each prompt–completion pair corresponds to one training unit. The stream contains 256 examples from each task, and each sequence contains examples from a single task. For each experimental seed, we shuffle the examples within each task and randomize the order of the resulting sequences. We sample without replacement, so every selected example occurs once in the stream. We consider

$$L \in \{32, 64, 128, 256\}. \tag{82}$$

As an i.i.d. reference, we shuffle all 1,024 examples across tasks and use one-example sequences with $L = 1$ and $\lambda = 1$.

**Local training and integration** We perform local adaptation using the completion-only SFT objective in Eq. (67). Prompt tokens are excluded from the loss, so the model is trained only on the expert-generated completion. We select the learning rate through a grid search on the i.i.d. reference and use the selected value of $5 \times 10^{-5}$ for all non-i.i.d. runs. We optimize the model using AdamW with zero weight decay, a minibatch size of one, and a constant learning-rate schedule. We use a maximum sequence length of 4,096 tokens.

After every sequence, we integrate both the model parameters and AdamW state. We use the same static coefficient throughout a run and consider

$$\lambda \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}. \tag{83}$$

We run both the non-i.i.d. experiments and the i.i.d. reference with seeds $0$, $1$, $2$, $3$, and $4$.

### D.6 REASONING WITH GRPO

**Data** For each reasoning task, we generate 4,096 training prompts and arrange them into 1,024 fixed minibatches of four prompts. The complete training set therefore contains 16,384 prompts across the four tasks.

**Streams and i.i.d. reference** Each four-prompt minibatch constitutes one training unit. In the non-i.i.d. experiments, every minibatch and sequence contains prompts from a single reasoning task. The task-balanced stream contains 1,024 minibatches per task, sampled without replacement. For each experimental seed, we randomize the order of the resulting sequences and avoid adjacent sequences from the same task whenever possible.

For the i.i.d. reference, we shuffle all 16,384 prompts across tasks, form mixed-task minibatches of four prompts, and use one-minibatch sequences with $L = 1$ and $\lambda = 1$.

**Local training and integration** We perform local adaptation using the DAPO-style clipped policy objective in Eq. (72). For each minibatch of four prompts, we sample eight on-policy rollouts per prompt, yielding a rollout minibatch of 32 completions. We use token-level importance ratios with clipping parameters $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$. Rewards are normalized within each group of eight rollouts, and the loss is normalized by the number of active completion tokens in the globally accumulated minibatch.

We optimize the model using AdamW with a learning rate of $10^{-6}$, zero weight decay, and a constant learning-rate schedule. During training, we generate completions with temperature 1, top-$p = 0.95$, top-$k = 20$, and a maximum of 2,048 new tokens. We use neither a reference-model KL penalty nor a reasoning-length reward. We consider

$$L \in \{256, 1024\} \quad \text{prompt minibatches}, \tag{84}$$

$$\lambda \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}. \tag{85}$$

After every sequence, we integrate both the model parameters and AdamW state. We use the shared reasoning evaluation protocol and run both the non-i.i.d. experiments and the i.i.d. reference with seeds 0, 1, and 5.

### D.7 PRETRAINING FROM RANDOM INITIALIZATION

**Model** We use the nanoGPT implementation of a decoder-only Transformer with the dimensions of GPT-2 small (Radford et al., 2019): 12 layers, 12 attention heads, and a hidden width of 768. Unlike the released GPT-2 checkpoint used for continual pretraining, this model is initialized randomly. The implementation uses a vocabulary padded to 50,304 entries, a context length of 1,027 tokens, no dropout, and no bias parameters. The resulting model has 123.6 million parameters. We perform training in BF16.

**Data and distributions** We select ten domains from the Pile Uncopyrighted dataset (Gao et al., 2020): *PubMed Central, ArXiv, GitHub, FreeLaw, StackExchange, USPTO Backgrounds, Gutenberg (PG-19), English Wikipedia, Hacker News, and DM Mathematics*.

Using seed 2357, we split each domain into disjoint training and validation sets through a deterministic document-level hash, assigning 99% of the documents to training and 1% to validation. We tokenize the documents using the GPT-2 tokenizer and append an end-of-sequence token after every document. We then pack the tokenized documents across document boundaries into blocks of 1,028 tokens, yielding 1,027 next-token prediction targets per block.

For each domain, we select $2^{20} = 1{,}048{,}576$ training blocks and 512 validation blocks. The selected blocks are fixed across all experimental seeds.

**Streams and joint-data reference** Each effective minibatch of 512 blocks constitutes one training unit and contains 525,824 next-token prediction targets. For each domain, we group the selected blocks into 2,048 minibatches. A complete stream therefore contains 20,480 optimizer updates and approximately 10.77 billion target tokens across the ten domains. Each non-i.i.d. minibatch contains blocks from a single domain, and each sequence consists of $L$ consecutive minibatches from that

domain. We randomize the order of the resulting sequences for each experimental seed and avoid adjacent sequences from the same domain whenever possible. Every selected minibatch occurs exactly once in each stream.

As a joint-data reference, we globally shuffle the same 10,485,760 training blocks and form mixed-domain minibatches with the same effective minibatch size. The reference uses the same random initialization, selected data, optimizer settings, and number of updates as the non-i.i.d. runs, but trains continuously without parameter or optimizer-state integration. It differs from the $L = 1$, $\lambda = 1$ condition, in which every minibatch remains domain-specific and an integration step is performed after every update.

**Local training and integration** We optimize the next-token language-modelling objective in Eq. (66). We select the learning rate through a grid search on the joint-data reference and use the selected value of $10^{-3}$ for all non-i.i.d. runs. We use AdamW with a weight decay of $0.1$, $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient-norm clipping at $1$, and a constant learning-rate schedule. We consider

$$L \in \{1, 64, 256, 1024, 2048\}, \tag{86}$$

$$\lambda \in \{0.2, 0.4, 0.6, 0.8, 1\}. \tag{87}$$

After every sequence, we integrate both the model parameters and AdamW state using the same static coefficient throughout a run.

**Evaluation and seeds** At each evaluation checkpoint, we evaluate one minibatch of 512 validation blocks from each domain and compute the macro-average BPT gain across the ten domains. For $L \geq 64$, we evaluate before local adaptation, after local adaptation but before integration, and after integration at every sequence. For $L = 1$ and for the joint-data reference, we evaluate every 64 optimizer updates.

Our primary trajectory statistic is the post-integration BPT gain, averaged first across domains, then across evaluation checkpoints within each run, and finally across seeds. We run both the non-i.i.d. experiments and the joint-data reference with seeds $0$, $1$, and $2$.

### D.8 Book-level continual pretraining

**Model** We perform full-parameter continual pretraining of SmolLM-360M (Ben Allal et al., 2024). Each run starts from the same pretrained checkpoint. Forward and backward passes use BF16 mixed precision, with parameters and optimizer moments in FP32.

**Books and adaptation sequences** For each paired data and training seed in $\{0, 1, 2\}$, we randomly permute the PG-19 training split (Rae et al., 2020) and select the first 200 books with at least two usable token blocks. All 24 configurations within a seed use the same books, order, and held-out partitions. Across the three seeds, the 600 book entries represent 597 distinct Gutenberg IDs. Books define the adaptation boundaries, while their order is seed-dependent rather than chronological.

We tokenize with the SmolLM-360M tokenizer without chat formatting or added special tokens and divide each book into non-overlapping blocks of at most 2,048 tokens, retaining partial blocks with at least two tokens. We randomly hold out $\lceil 0.1 N_b \rceil$ of a book's $N_b$ usable blocks and process the remainder once in text order, without replay or cross-book packing. With four blocks per optimizer step, the number of updates before integration is $\lceil N_b^{\text{train}}/4 \rceil$.

**Local training and integration** We minimize next-token cross-entropy using AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, no weight decay, and gradient-norm clipping at $1$. The global batch contains four blocks, one per GPU, without gradient accumulation; padding and dummy examples are excluded from the token-normalized loss. We use constant learning rates $\eta \in \{1, 2, 3, 4\} \times 10^{-5}$ and $\lambda \in \{0.4, 0.6, 0.8, 1\}$, integrating parameters and optimizer state after each book according to Eqs. (3) and (63) to (65). For each partial-integration setting, we include full integration at learning rate $\lambda\eta$. Reusing shared baselines gives 24 configurations per seed and 72 runs in total.

**Evaluation and seeds** We evaluate the continuing model's general capabilities after each integration on two fixed sampled blocks from each of 50 separate validation books, identical across configurations and seeds. The primary score is BPT reduction relative to initialization, averaged equally over validation books and the 200 integration boundaries. We report the mean and SEM across the three seeds.

**Length and exposure** Full-book lengths range from 3,011 to 1,435,195 tokens. Median lengths are 80.9k–83.2k tokens across seeds. The resulting adaptation sequences contain 1–158 optimizer steps, with medians of 9–9.5 steps. Figure 10 and Table 3 report the distributions and per-run exposure; totals are not multiplied by the number of learning-rate and integration configurations.

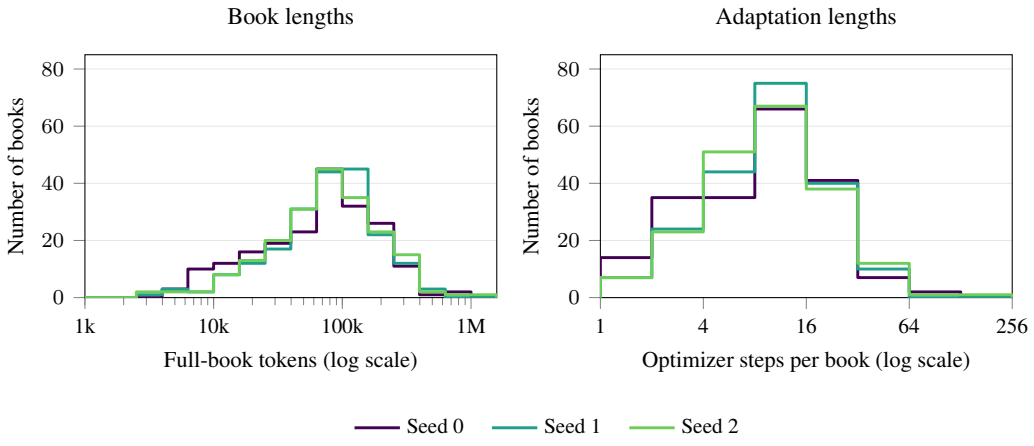


Figure 10: PG-19 book lengths and adaptation lengths for 200 books per seed. Outline histograms use common logarithmically spaced bins within each panel. Full-book token counts include the held-out portion; optimizer steps count only training updates before integration.

Table 3: Book lengths and training exposure per run. k and M denote thousands and millions of tokens. Training targets exclude held-out blocks, the first token of each block, and padding. P90 uses nearest rank.

| Seed | Full-book tokens (k) | | Training targets (M) | Steps per book | | Total steps |
|---|---|---|---|---|---|---|
| | Median | P90 | | Median | Range | |
| 0 | 80.9 | 221.4 | 18.49 | 9 | 1–91 | 2,353 |
| 1 | 81.1 | 219.0 | 18.98 | 9 | 1–57 | 2,410 |
| 2 | 83.2 | 242.1 | 20.41 | 9.5 | 1–158 | 2,585 |

**Catalog categories** All 597 distinct IDs match the supplied Project Gutenberg catalog snapshot.[1] Table 4 reports its eight most frequent category labels across the 600 seed/book entries. Labels overlap and are descriptive catalog annotations, not mutually exclusive genres or training-task labels. The catalog language field is English-only for 597 entries; two carry English plus another language and one is labeled Italian. These are metadata annotations rather than language detection on the training text.

[1] `https://www.gutenberg.org/ebooks/offline_catalogs.html`.

Table 4: Most frequent Gutenberg categories. Entries are percentages of the 200 books in each seed; overlapping labels need not sum to 100%.

| Category | Seed 0 | Seed 1 | Seed 2 |
|---|---|---|---|
| Novels | 32.5 | 30.0 | 32.0 |
| British Literature | 20.0 | 18.0 | 19.5 |
| American Literature | 10.0 | 10.0 | 14.0 |
| History - Modern (1750+) | 10.5 | 10.5 | 13.0 |
| Adventure | 11.5 | 12.0 | 9.5 |
| Biographies | 9.0 | 8.5 | 12.5 |
| Children & Young Adult Reading | 10.0 | 9.0 | 10.0 |
| Historical Novels | 10.5 | 6.5 | 10.5 |

### D.9 Controlled long-SFT interpolation

**Data and training** We shuffle MetaMathQA with seed 42, remove examples with empty queries or responses, and select the first 100,000 valid examples. A deterministic 99/1 split provides 99,000 training examples and 1,000 held-out examples; the held-out examples are not used for training or checkpoint selection. Each example is formatted with the native SmolLM2 chat template and the system message "You are a helpful AI assistant that solves math problems step by step." The MetaMathQA response is retained verbatim apart from surrounding whitespace, and the loss is applied only to assistant-response tokens.

We fine-tune all parameters for four epochs using fused AdamW with a peak learning rate of $10^{-5}$, cosine decay, a warmup ratio of $0.05$, weight decay of $0.01$, and gradient clipping at a global norm of $1.0$. Training uses a maximum sequence length of 2,048 without packing, BF16 precision, and an effective global batch size of 16. Training is distributed across four NVIDIA GH200 GPUs using data parallelism.

**Checkpoints and interpolation** We analyze checkpoints after 50, 100, 250, 500, 1,000, 1,500, 2,000, 3,000, 5,000, 7,500, 10,000, 12,500, 15,000, 17,500, and 20,000 optimizer steps. For checkpoint parameters $\theta_s$ and initial parameters $\theta_0$, we construct $\theta_{s,\lambda} = (1-\lambda)\theta_0 + \lambda\theta_s$ for $\lambda \in \{0, .2, .4, .6, .8, 1\}$. This produces 64 interior interpolants in addition to the initial model and SFT endpoints. Floating-point tensors are interpolated elementwise in BF16, and all models share the same architecture, tokenizer, vocabulary, and chat template.

**Evaluation** We evaluate the complete GSM8K test set using zero-shot greedy generation with one response per problem. Evaluation uses the training-time system message and native chat template, a maximum generation length of 1,024 tokens, and a 2,048-token context. Predictions are scored with `math_verify`, which parses and symbolically compares the generated answer with the numerical reference following the dataset's `####` delimiter. We report mean verification accuracy over all test examples.

### D.10 Qwen2.5–Coder diagnostic

In this experiment, we test whether linear parameter connectivity is preserved after a much larger adaptation. This is a diagnostic between released checkpoints, not a continual-training experiment.

**Models and parameter path** We consider the 1.5B, 3B, and 7B parameter sizes of the Qwen2.5 family. For each scale, we use the released Qwen2.5 base checkpoint as the generalist model and the Qwen2.5-Coder checkpoint as the specialist (Yang et al., 2024; Hui et al., 2024). The Coder lineage was pretrained on 5.2T tokens at file level and approximately 300B additional tokens at repository level. We construct the direct linear path

$$\theta_\lambda = (1-\lambda)\theta_{\text{base}} + \lambda\theta_{\text{coder}}, \qquad \lambda \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}, \tag{88}$$

where $\lambda = 0$ corresponds to the base checkpoint and $\lambda = 1$ to the Coder checkpoint. This experiment requires no additional training or optimizer-state integration.

**Evaluation** We evaluate each checkpoint using vLLM. As the targeted benchmark, we use all 500 problems from the MBPP test set (Austin et al., 2021) with the three-shot harness prompt and report pass@1 based on executable tests.

To measure retention, we evaluate the complete MMLU (Hendrycks et al., 2021), ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2020) evaluation sets. We define the non-targeted score as the unweighted mean of MMLU accuracy, normalized ARC-Challenge accuracy, normalized HellaSwag accuracy, and WinoGrande accuracy. We use the same evaluation configuration at every point along the interpolation path.

### D.11 Compute resources and recorded runtime

All experiments, including PG-19, used shared HPC nodes with four NVIDIA GH200 GPUs (96 GB HBM3 each), 288 Arm CPU cores, and 512 GB nominal host memory.[2]

Table 5: Recorded compute for the six training settings. Hours/run gives the runtime range; GPU-hours sums runtime multiplied by allocated GPUs. Timings include evaluation.

| Setting | GPUs/run | Timed runs | Hours/run | GPU-hours |
|---|---|---|---|---|
| Continual pretraining | 4 | 72 | 0.11–0.55 | 79.85 |
| Persona SDPO | 2 | 84 | 0.91–2.94 | 292.57 |
| Reasoning SFT | 4 | 125 | 0.29–3.58 | 540.36 |
| Reasoning GRPO | 4 | 39 | 4.53–8.51 | 896.76 |
| Pretraining from scratch | 4 | 78 | 1.94–2.59 | 663.71 |
| PG-19 continual pretraining | 4 | 72 | 0.49–1.09 | 202.87 |
| Recorded subtotal | | 470 | | 2,676.13 |

## E Extended results

This section provides additional integration trade-offs, long-SFT interpolation results, and the complete pretraining-scale interpolation diagnostic.

### E.1 Additional integration trade-offs

**Immediate effect of task-vector scaling** To isolate the immediate effect of applying $\lambda$, we compare the continuing model $\theta_{t+1}$ with the endpoint $\theta_t^{(L)}$. For each distribution, define the score change caused by scaling as

$$\delta_{t,k}^{\text{scale}} = \text{score}(\theta_{t+1}; k) - \text{score}(\theta_t^{(L)}; k). \tag{89}$$

The targeted-distribution change is $\delta_{t,k_t}^{\text{scale}}$, while the remaining $\delta_{t,k}^{\text{scale}}$ measure score changes on non-targeted distributions. The corresponding change in average performance decomposes as

$$\text{ScaleEffect}_t = \frac{1}{K}\sum_{k=1}^{K}\delta_{t,k}^{\text{scale}} = \underbrace{\frac{1}{K}\sum_{k\neq k_t}\delta_{t,k}^{\text{scale}}}_{\text{non-targeted contribution}} + \underbrace{\frac{1}{K}\delta_{t,k_t}^{\text{scale}}}_{\text{targeted contribution}}. \tag{90}$$

Here, $\delta_{t,k_t}^{\text{scale}} < 0$ means that scaling removes part of the targeted-distribution gain, whereas positive non-targeted changes indicate recovered performance on the other distributions. In our plots, we call the non-targeted contribution *non-targeted recovery* and the negative targeted contribution *targeted loss*. Thus, $\text{ScaleEffect}_t$ equals non-targeted recovery minus targeted loss. Positive values indicate recovered non-targeted performance and lost targeted-distribution performance, respectively.

Figure 11 plots non-targeted recovery against targeted loss, as defined by Eq. (90). Points below the diagonal improve average performance over full integration; above it, targeted loss exceeds non-targeted recovery.

[2] https://www.cscs.ch/computers/alps.

For SFT, scaling recovers substantial non-targeted performance, placing several intermediate coefficients below the diagonal. GRPO recovers little, and most points lie above it. This contrast may reflect off-policy SFT versus on-policy GRPO: on-policy adaptation appears to disrupt non-targeted performance less, leaving less to recover. This interpretation is consistent with *RL's Razor*, which finds that on-policy RL forgets less than SFT by remaining closer in KL divergence to the base policy (Shenfeld et al., 2026). These averages capture only the immediate effect, not downstream effects on later checkpoints.

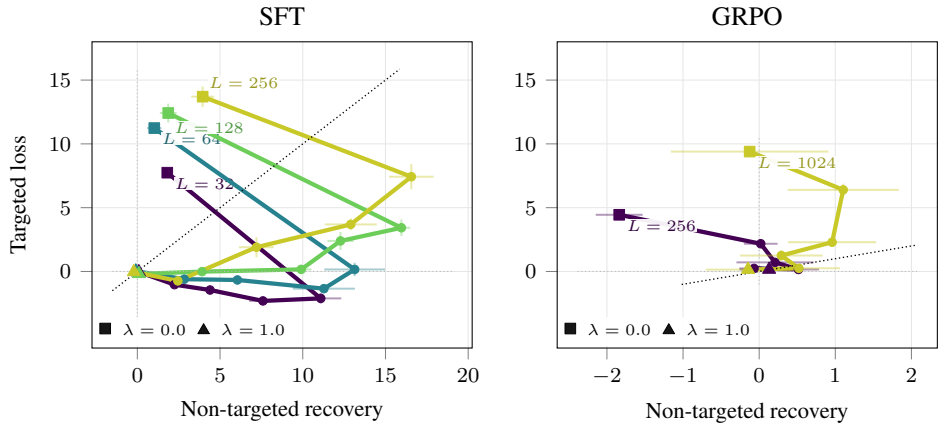


Figure 11: Immediate effect of task-vector scaling for SFT (left) and GRPO (right). The dotted diagonal marks zero change in average performance relative to full integration.

Figure 12 extends this analysis to continual pretraining and persona SDPO. In continual pretraining, partial-integration points lie below the zero-change boundary for every evaluated $L$, indicating that non-targeted recovery outweighs the loss on the targeted distribution. Persona SDPO is less monotonic: some intermediate coefficients improve targeted and non-targeted style scores simultaneously, whereas very small coefficients can reduce non-targeted performance. This pattern is consistent with the on-policy feedback updates improving multiple measured style traits rather than inducing a strict adaptation–retention trade-off.

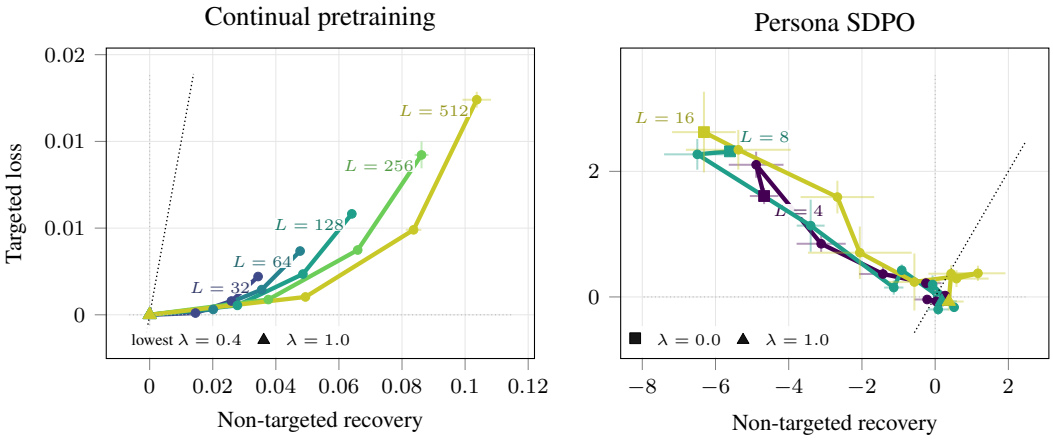


Figure 12: Task-count-normalized integration trade-off for continual pretraining and persona SDPO. The horizontal axis is the combined non-targeted recovery contribution and the vertical axis is the targeted-loss contribution. The dotted diagonal is the zero-change boundary. Circles mark partial integration ($0 < \lambda < 1$), squares mark $\lambda = 0$ where evaluated, and triangles mark $\lambda = 1$.

**Pretraining from random initialization** Figure 13 shows the immediate effect of scaling during pretraining from random initialization. Very small coefficients reduce even non-targeted performance

for $L \in \{1024, 2048\}$, yielding negative non-targeted recovery. This pattern is consistent with task vectors containing generally useful representation learning in addition to distribution-specific adaptation; retaining too little of the vector can therefore impede learning across distributions.

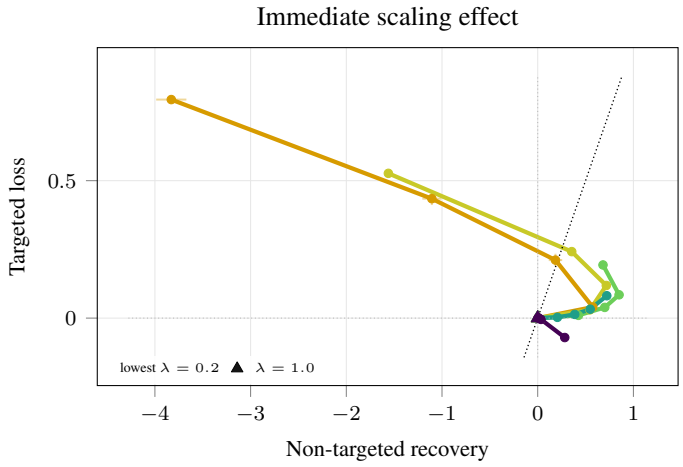


Figure 13: Immediate effect of task-vector scaling during pretraining from random initialization. Non-targeted recovery and targeted loss are defined in Eq. (90); their difference gives the change in average performance. The dotted diagonal marks zero change. Colors denote sequence lengths as in Fig. 7; circles mark partial integration and triangles mark $\lambda = 1$. Error bars show SEM across seeds.

### E.2 Long-SFT interpolation

Task-vector scaling relies on intermediate parameters remaining useful as local adaptation progresses. We examine whether longer adaptation trajectories introduce performance barriers between the starting model and the adapted endpoint. Figure 14 reports GSM8K accuracy along interpolations between the initial SmolLM2-1.7B-Instruct model and checkpoints from a controlled MetaMathQA SFT trajectory, described in Appendix D.9. Across the evaluated checkpoints, accuracy varies smoothly with $\lambda$, without a pronounced interior performance barrier, so intermediate task-vector scaling remains effective across the tested adaptation lengths.

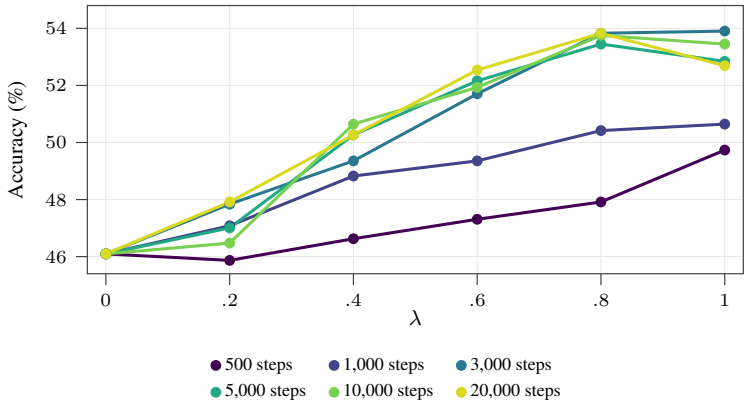


Figure 14: GSM8K accuracy along interpolations between the initial model and checkpoints obtained after different numbers of MetaMathQA SFT steps.

### E.3 QWEN2.5–CODER DIAGNOSTIC

To test the limits of direct interpolation beyond the controlled long-SFT trajectory, we evaluate the released Qwen2.5 base and Qwen2.5-Coder checkpoints described in Appendix D.10. Figure 15 reports targeted and non-targeted performance along this path at the 1.5B, 3B, and 7B parameter scales.

Interior interpolation points perform worse than both endpoints on targeted and non-targeted scores at all three model scales. For example, at 7B parameters, MBPP falls from $63.4\%$ at the base checkpoint and $68.8\%$ at the Coder checkpoint to $18.2\%$ at $\lambda = 0.4$; the non-targeted score falls from $68.79\%$ and $65.10\%$ at the endpoints to $43.84\%$. These results mark a pretraining-scale boundary to the well-behaved interpolation observed in Section 5.5.

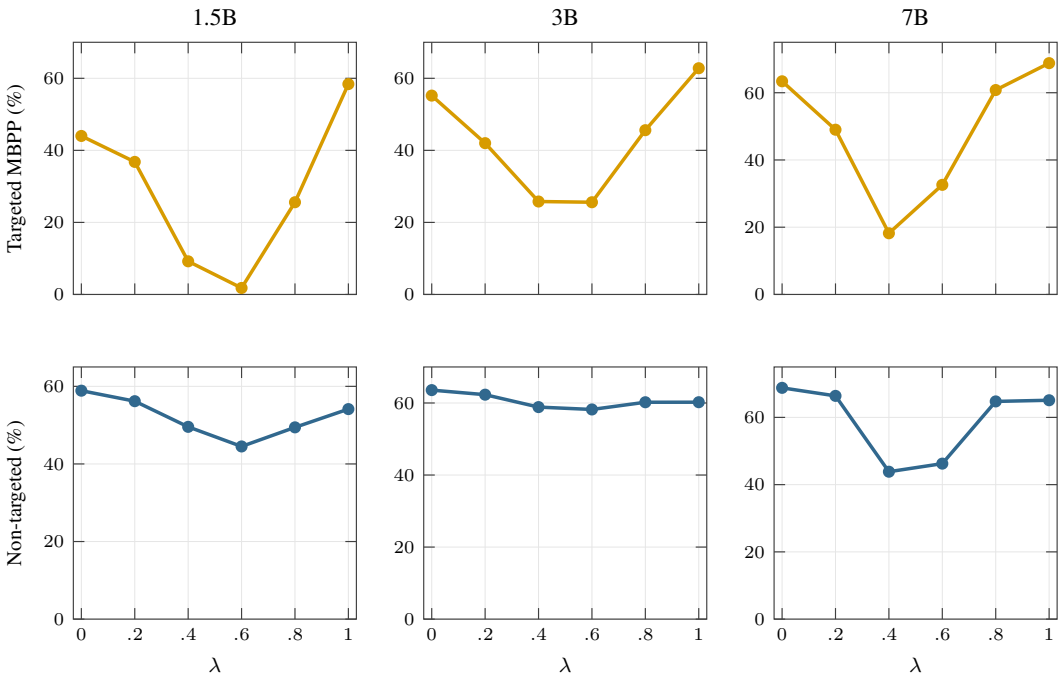


Figure 15: Direct interpolation between released Qwen2.5 base checkpoints and their Qwen2.5-Coder descendants. Columns show parameter scale. The top row reports targeted MBPP pass@1, while the bottom row reports the unweighted mean of full MMLU, normalized ARC-Challenge, normalized HellaSwag, and WinoGrande accuracy.