

Midtraining Bridges Pretraining and Posttraining Distributions

Emmy Liu¹ Graham Neubig¹ Chenyan Xiong¹

Abstract

Midtraining, the practice of mixing specialized data with more general pretraining data in an intermediate training phase, has become widespread in language model development, yet there is little understanding of what makes it effective. We propose that midtraining functions as distributional bridging by providing better initialization for post-training. We conduct controlled pretraining experiments, and find that midtraining benefits are largest for domains distant from general pretraining data, such as code and math, and scale with the proximity advantage the midtraining data provides toward the target distribution. In these domains, midtraining consistently outperforms continued pretraining on specialized data alone both in-domain and in terms of mitigating forgetting. We further conduct an investigation on the starting time and mixture weight of midtraining data, using code as a case study, and find that time of introduction and mixture weight interact strongly such that early introduction of specialized data is amenable to high mixture weights, while late introduction requires lower ones. This suggests that late introduction of specialized data outside a plasticity window cannot be compensated for by increasing data mixtures later in training. Beyond midtraining itself, this suggests that distributional transitions between any training phases may benefit from similar bridging strategies.¹

1. Introduction

The success of large language models has mostly been driven by scaling model and data size. Though many interventions seem promising, they may wash out at scale. Therefore, when methodological interventions are simple yet widely adopted across model scales, they merit attention.

¹Language Technologies Institute, Carnegie Mellon University, USA. Correspondence to: Emmy Liu <emmy@cmu.edu>.

Preprint. February 3, 2026.

¹Data and code are available at <https://anonymous.4open.science/r/midtraining-E5D8/>.

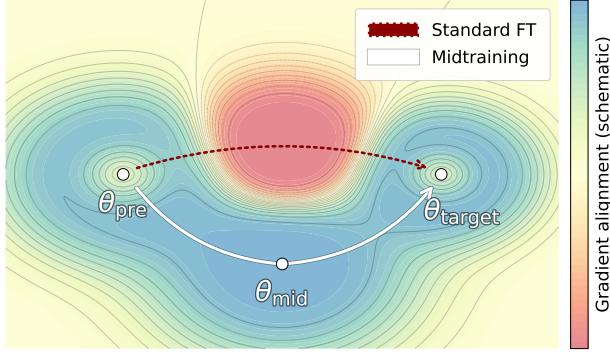


Figure 1. Schematic optimization landscape, where contours indicate gradient conflict between pretraining and posttraining objectives. Standard pretraining→SFT follows the red path, while pretraining→midtraining→SFT follows the white path. Midtraining shifts the initialization so SFT can approach the target while avoiding high-conflict regions.

One such intervention is *midtraining*: breaking pretraining into two or more stages in which the latter stages incorporate higher-quality data from specialized domains such as mathematics and coding, as well as instruction-formatted data (Hu et al., 2024b; Dubey et al., 2024; OLMo Team et al., 2025). While widely adopted, it is often treated as a heuristic “cooldown” phase, with surprisingly little systematic study of its underlying mechanics or optimal design (Wang et al., 2025).

This raises key questions: is midtraining simply a form of teaching to the test by learning the target distribution, or does it serve a more distinct role in the training trajectory? When is it expected to help performance in-domain, and does it have an impact on forgetting of the pretraining distribution? How does it compare to the similar concept of continued pretraining? To answer these questions, we conduct the first systematic investigation of midtraining, controlling for data domains, mixture ratios, and the timing of this phase relative to overall training schedule.

Our results that midtraining appears to act as a distributional bridge, smoothing the optimization path from general pretraining data to specialized domains. By systematically varying training conditions, we identify specific conditions under which midtraining is most effective:

- **Midtraining yields the largest gains on domains that**

are “distant” from the general pretraining distribution, such as mathematics and code. In these high-shift regimes, midtraining appears to mitigate the gradient conflicts that typically hamper standard fine-tuning.

- **Midtraining reduces catastrophic forgetting** compared to both direct fine-tuning and naive continued pretraining. This challenges the “memorization” hypothesis, as the method preserves prior knowledge better than simply training on the target data at the end.
- **The timing of data introduction is often more impactful than the mixture weight itself.** This aligns with a “Plasticity Window” hypothesis, suggesting that midtraining is most effective when applied while the model’s representations remain sufficiently malleable to adjust to the new distribution without rigidity.

Taken together, these findings offer a more nuanced view of midtraining: not merely as a final polishing step, but as a geometric intervention that when timed correctly allows models to specialize in distant domains while mitigating forgetting of general knowledge.

2. Preliminaries

In this section, we define what we refer to as midtraining throughout this paper. While this term has been used colloquially by model developers, it lacks a standard definition, so we establish our working definition for clarity.

2.1. Training Sequence Definitions

Language model training can be viewed as a sequence of phases $S = \{D_i, J_i\}_{i=0}^N$, where parameters θ_i are initialized from training on data up to phase $i-1$. Standard LM training typically consists of first **pretraining** on a massive, diverse corpus D_{pre} , followed by **posttraining** on a target dataset D_{target} , where often D_{target} is orders of magnitude smaller and usually on a narrower topic semantically.

We define **midtraining** as any intermediate phase between training stages, in this case between pretraining and post-training. Midtraining data is typically more specialized than general pretraining data, often including domain-specific content (code, math) and instruction-formatted data, while maintaining a mixture with general pretraining data. Typically, midtraining is a longer phase compared to fine-tuning, but shorter than the preceding pretraining phase, $|D_{\text{pre}}| > |D_{\text{mid}}| > |D_{\text{target}}|$. There can potentially be multiple midtraining phases as well in the case of multi-stage pretraining curricula, but we focus on one-stage midtraining in this paper.

2.2. Relationship to Curriculum Learning and Continued Pretraining

Curriculum learning The original definition of curriculum learning focused on gradually increasing the difficulty or diversity of training examples throughout the course of training (Elman, 1993; Bengio et al., 2009). However, this term has evolved to generally mean any strategic ordering of training data (Soviany et al., 2022). Midtraining can be viewed as a coarse-grained distributional curriculum; instead of ordering individual examples, it orders data distributions across discrete training phases.

Continued pretraining Continued pretraining adapts a pretrained model by training further on domain-specific data, typically with a full shift to the target distribution (Gururangan et al., 2020; Beltagy et al., 2020). This can improve in-domain performance but risks degrading general capabilities. Midtraining differs in that it retains a mixture with general pretraining data during the intermediate phase. In our setup, continued pretraining is the limiting case where the mixture weight on general pretraining data is zero. In practice, both are often implemented as additional next-token-prediction training, differing mainly in the degree of distribution shift and associated schedule/optimizer choices.

2.3. Theoretical Analysis

In the previous section, we defined midtraining as an intermediate phase that mixes general pretraining data with a specialized distribution before posttraining. Here, we give a simple theoretical sketch that formalizes the core intuition behind our empirical results: midtraining acts primarily through the initialization for posttraining, and this can simultaneously (i) improve in-domain posttraining loss and (ii) mitigate forgetting on the pretraining distribution. Full derivations appear in Appendix A.

Setup. Let $J_P(\theta)$ denote the population loss on the pre-training distribution P , and $J_T(\theta)$ the loss on the posttraining/SFT distribution T . Posttraining runs K steps of gradient descent on J_T starting from an initialization θ_0 :

$$\theta_{k+1} = \theta_k - \eta \nabla J_T(\theta_k), \quad k = 0, \dots, K-1, \quad (1)$$

and we measure forgetting by the increase in pretraining loss, $\Delta_P(K) := J_P(\theta_K) - J_P(\theta_0)$. Midtraining affects posttraining only through the resulting initialization, which we write as $\theta_0 = \theta_0(t, w)$ to emphasize its dependence on the midtraining start time t and mixture weight w .

Forgetting decomposition (smoothness sketch). Assume J_P is L_P -smooth. Consider K steps of posttraining GD on J_T , $\theta_{t+1} = \theta_t - \eta \nabla J_T(\theta_t)$. Applying the standard smoothness upper bound to J_P at θ_t with step

$\theta_{t+1} - \theta_t = -\eta \nabla J_T(\theta_t)$ gives the one-step inequality

$$J_P(\theta_{t+1}) - J_P(\theta_t) \leq -\eta \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle + \frac{L_P \eta^2}{2} \|\nabla J_T(\theta_t)\|^2. \quad (2)$$

Summing (2) over $t = 0, \dots, K-1$ yields a telescoping sum:

$$\begin{aligned} \Delta_P(K) &:= J_P(\theta_K) - J_P(\theta_0), \\ \Delta_P(K) &\leq -\eta \underbrace{\sum_{t=0}^{K-1} \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle}_{\text{(A) alignment / conflict along the posttraining path}} \\ &\quad + \underbrace{\frac{L_P \eta^2}{2} \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2}_{\text{(B) posttraining “energy” (squared-gradient) term}}. \end{aligned} \quad (3)$$

Relating (B) to posttraining progress. If J_T is L_T -smooth and $\eta \leq 1/L_T$, a standard GD descent inequality gives $J_T(\theta_{t+1}) \leq J_T(\theta_t) - \frac{\eta}{2} \|\nabla J_T(\theta_t)\|^2$. Summing over t yields

$$\begin{aligned} \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2 &\leq \frac{2}{\eta} (J_T(\theta_0) - J_T(\theta_K)) \\ &\leq \frac{2}{\eta} (J_T(\theta_0) - J_T^*), \end{aligned} \quad (4)$$

where $J_T^* := \inf_{\theta} J_T(\theta)$.

Finally, we can substitute back to get the forgetting bound:

$$\begin{aligned} \Delta_P(K) &\leq -\eta \underbrace{\sum_{t=0}^{K-1} \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle}_{\text{(A) alignment term}} \\ &\quad + \underbrace{L_P \eta (J_T(\theta_0) - J_T^*)}_{\text{(B) effort term}}. \end{aligned} \quad (5)$$

Connection to midtraining. Because midtraining changes only the initialization $\theta_0 = \theta_0(t, w)$, it can change the upper bound on forgetting by providing an initialization which has to do less work initially ($J_T(\theta_0) - J_T^*$ smaller).

3. Experimental Setting

Having defined midtraining as an intermediate phase between pretraining and posttraining, we next specify the controlled experiments we use to study this training phase. Across our experiments, we keep the model family and posttraining procedure fixed and vary the conditions of mid-training. We organize this section around four key research questions, which we introduce one by one along results.

Midtrain mix	Num. Tokens (B)	Sources
Starcoder	196	(Li et al., 2023)
Math	12	(Yue et al., 2023; Toshniwal et al., 2024)
FLAN	3.5	(Wei et al., 2022)
KnowledgeQA	9.6	(Hu et al., 2024a)
DCLM	51	(Li et al., 2024b)

Table 1. Midtraining mixes used in our experiments and dataset(s) from which they were derived.

3.1. Training Setup

Pretraining We pretrain models from the Pythia family ranging in size from 70M-1B parameters on C4 web data (Raffel et al., 2020; Biderman et al., 2023). In all cases, we train for 128B tokens (approx. 61k steps) with a cosine learning rate schedule with a maximum learning rate of 3e-4 and the AdamW optimizer (Loshchilov & Hutter, 2019). We chose to fix the training budget at a point past Chinchilla-optimality for all models (Hoffmann et al., 2022), in order to ensure that models have stabilized by the point at which midtraining data has been introduced, at least for later insertion points of midtraining data. We describe the exact training setup in Appendix B.

Midtraining We use five midtraining mixtures spanning popular domains: code (Starcoder), math, instructions (FLAN), general knowledge/QA, and high-quality web data (DCLM). Table 1 details each mixture’s composition and sources. All mixtures are introduced at varying start points (Starcoder: 6k steps, Math: 20k steps, others: 40k steps) based on data availability to prevent repetition. We compare against a control condition continuing C4 pretraining for the same number of tokens, keeping all other training details identical.

Starcoder (code) Our code mixture is a subset of the Starcoder pretraining dataset (Li et al., 2023), which contains code in many languages. Note that we use code from all languages, rather than Python.

Math The math mixture combines mathematical reasoning problems from the MAMmoTH (Yue et al., 2023) and OpenMathInstruct (Toshniwal et al., 2024) datasets, featuring step-by-step explanations.

FLAN (instructions) Our instruction-formatted data comes from a processed version of the FLAN collection, which includes diverse task instructions and responses across natural language tasks (Wei et al., 2022).

KnowledgeQA (general knowledge and QA) The KnowledgeQA mixture is taken from Hu et al. (2024b)’s midtraining mix, and focuses on general knowledge and dialogue. However, to distinguish the midtraining mixes fur-

ther, the StackOverflow portion of this dataset is removed.

DCLM (high-quality web) Our high-quality web data is a subset of the DCLM pretraining dataset, representing web content with improved quality filtering compared to C4 (Li et al., 2024b).

Downstream Evaluation We fine-tune models on the datasets GSM8k (Cobbe et al., 2021), SciQ (Welbl et al., 2017), CodeSearchNet-Python (Husain et al., 2019), and LIMA (Zhou et al., 2023) – chosen to span the domains covered by our midtraining mixtures. This allows us to test cases where the midtraining mixture is aligned or misaligned with the SFT dataset. We used standard language model supervised fine-tuning for all datasets. For information on the posttraining setup, see Appendix C.

Catastrophic Forgetting Evaluation A key concern with supervised fine-tuning is whether introducing specialized data causes models to forget general capabilities acquired during pretraining. We measure catastrophic forgetting by evaluating cross-entropy loss on the original pretraining distribution by measuring loss on held-out C4 data. This approach follows established practices for measuring forgetting in language models (Luo et al., 2024; Kemker et al., 2018; Li et al., 2024a).

Proximity advantage. To quantify whether a midtraining mixture moves the training distribution toward a target SFT dataset, we compute a token-level proximity score $\text{prox}(\cdot, \cdot)$ between corpora using unigram token statistics under the model tokenizer (example in Figure 2, full in Appendix D). Given a target dataset T and a midtraining mixture M , we define the *proximity advantage* of M relative to continuing pretraining on C4 as

$$\text{PA}(M \rightarrow T) = \text{prox}(M, T) - \text{prox}(\text{C4}, T). \quad (6)$$

Positive PA indicates that M is closer to T than C4 at the token level. While our theory is stated in terms of optimization quantities (e.g., gradient alignment), PA provides an inexpensive, model-agnostic diagnostic of distribution shift.

4. Which downstream tasks benefit most from midtraining?

We begin by asking **where** midtraining is most effective. We evaluate all combinations of midtraining mixtures and SFT targets, reporting (i) target-domain validation loss after SFT (adaptation) and (ii) C4 validation loss after SFT (forgetting). We average results over 5 seeds after hyperparameter search for each checkpoint.

We find that across model sizes, midtraining benefits are highly domain-specific: specialization on code yields the

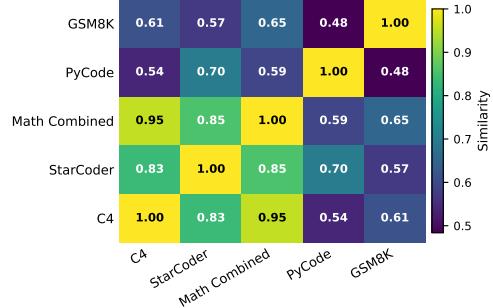


Figure 2. Example similarity matrix between pre/midtrain and posttraining datasets. For the complete matrix, see Appendix D.

largest gains on code tasks, while math-focused midtraining helps mathematical-reasoning tasks. Mismatched midtraining provides minimal benefit, and general instruction mixes (e.g., FLAN) produce little improvement. Full per-dataset results and numerical comparisons are reported in Table 2 and Appendix E. Interestingly, in this setting, in-domain improvements and C4 retention are strongly aligned (Pearson $r = 0.64$, $p \approx 3.7 \times 10^{-12}$).

Finding 1. Midtraining benefits are highly domain-specific. Code-focused midtraining (Starcoder) yields large gains on coding tasks, and math-focused midtraining improves mathematical reasoning. Mismatched domains provide little benefit, and broad instruction data (FLAN) also shows minimal effect.

5. What data is most effective for midtraining?

Having established that midtraining effects are domain-specific, we now ask: *what determines the strength of these domain-specific effects?* Across midtraining-target pairs, we see improvements ranging from negligible (e.g. FLAN → coding) to strong (e.g. Starcoder → coding). We consider two candidate explanations: (i) whether the midtraining distribution is “closer” to the target than C4 (distributional bridging), and (ii) whether retaining some general data is necessary compared to switching fully to specialized data.

5.1. Proximity and Bridging Effects

To understand why some midtraining mixtures are effective, we test the hypothesis that good midtraining data *bridges* the distributional gap between pretraining (C4) and the target SFT dataset. Concretely, we use the proximity advantage $\text{PA}(M \rightarrow T)$ defined in Equation 6, the increase in token-level proximity to the target achieved by midtraining on mixture M relative to continuing on C4 (Appendix D). Results are shown in Figure 3.

We find a clear relationship between proximity advantage

Table 2. SFT and C4 validation losses for the 1B model across downstream datasets and midtraining mixtures (5 seeds per SFT dataset). Bold indicates best within each dataset. Percentages denote the specialized data proportion mixed with C4 during midtraining. Parentheses show Δ relative to the C4-only baseline within each dataset (negative is better).

Dataset	Midtrain Mix	Validation Loss	
		SFT	C4
Pycode	C4	2.174 (+0.000)	3.075 (+0.000)
	Starcoder (20%)	1.888 (-0.286)	3.070 (-0.005)
	Math (12%)	2.175 (+0.000)	3.057 (-0.018)
	FLAN (5%)	2.174 (+0.000)	3.083 (+0.008)
	KnowledgeQA (20%)	2.174 (+0.000)	3.104 (+0.029)
	DCLM (20%)	2.175 (+0.001)	3.106 (+0.031)
GSM8K	C4	0.942 (+0.000)	3.134 (+0.000)
	Starcoder (20%)	0.927 (-0.015)	3.158 (+0.024)
	Math (12%)	0.851 (-0.091)	3.018 (-0.116)
	FLAN (5%)	0.942 (+0.000)	3.135 (+0.001)
	KnowledgeQA (20%)	0.943 (+0.001)	3.136 (+0.002)
	DCLM (20%)	0.943 (+0.001)	3.137 (+0.003)
LIMA	C4	3.316 (+0.000)	2.900 (+0.000)
	Starcoder (20%)	3.315 (-0.001)	2.930 (+0.030)
	Math (12%)	3.310 (-0.006)	2.893 (-0.008)
	FLAN (5%)	3.316 (-0.001)	2.900 (+0.000)
	KnowledgeQA (20%)	3.315 (-0.002)	2.900 (+0.000)
	DCLM (20%)	3.314 (-0.002)	2.900 (-0.001)
SciQ	C4	1.858 (+0.000)	3.201 (+0.000)
	Starcoder (20%)	1.873 (+0.015)	3.245 (+0.044)
	Math (12%)	1.876 (+0.018)	3.182 (-0.019)
	FLAN (5%)	1.856 (-0.002)	3.192 (-0.009)
	KnowledgeQA (20%)	1.856 (-0.002)	3.196 (-0.005)
	DCLM (20%)	1.857 (-0.001)	3.201 (+0.000)

and downstream performance improvements across model sizes. The correlations are particularly strong for smaller models ($r = 0.869$, $p < 0.001$ for 70m), suggesting that effective midtraining data serves as a distributional stepping stone from general pretraining to specialized target domains. This bridging effect appears to be most beneficial when the gap between pretraining and target distributions is large, consistent with our hypothesis that midtraining helps models adapt gradually rather than requiring abrupt distributional shifts during fine-tuning.

Finding 2. Midtraining gains are well-predicted by a simple proximity advantage metric: mixtures that are closer to the target than C4 (in terms of token distributions) yield larger improvements, especially for smaller models.

5.2. Midtraining vs. Continued Pretraining

Our results so far suggest that effective midtraining data serves as a bridge between general pretraining and specialized posttraining data. However, a question that follows is why midtraining is necessary: continued pretraining on domain-specific data also aims to adapt the model toward a target domain. Why not simply pretrain normally and then switch to domain-specific data entirely?

To examine this, we compare the effect of midtraining with continued pretraining in which the mixture weight switches to 100% specialized data. For code, we compare the default Starcoder midtraining mix (20% mixture weight, starting from 12.6B tokens) with 100% Starcoder data starting from 83B tokens. For math, we compare the math midtraining mix with 100% math data starting from 105B tokens.²

Results in Table 3 show that midtraining consistently outperforms continued pretraining across both domains and model sizes for both in-domain performance and C4 retention after fine-tuning. As this pattern holds for both code and math domains, this suggests that maintaining some general pretraining data is useful during domain adaptation, even for models specialized for a specific domain. This supports our intuition gained from prior sections that domain adaptation benefits from gradual distributional shifts at the token level rather than abrupt changes.

Finding 3. Maintaining a mixture with general data in midtraining is preferable to continued pretraining on specialized data alone.

²The different starting points are due to data availability, to ensure the midtraining mix does not repeat.

Midtraining Bridges Pretraining and Posttraining Distributions

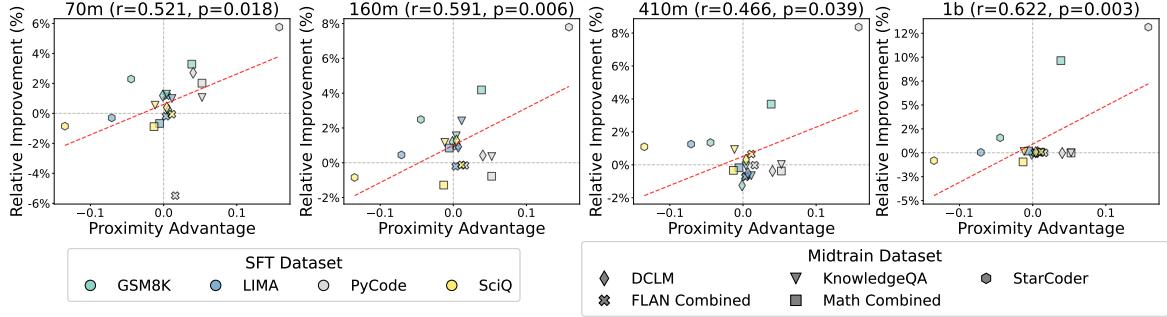


Figure 3. Relationship between proximity advantage and midtraining performance improvements for pairs of midtraining and SFT datasets. Each data point represents a (midtrain, SFT) pair, where the color indicates the SFT dataset and shape represents midtrain dataset. Proximity advantage ($\text{dist}(\text{C4}, \text{SFT}) - \text{dist}(\text{midtrain}, \text{SFT})$) indicates how much closer midtraining data brings the model to the target SFT dataset compared to the base pretraining data. Proximity advantage pairs near zero are greyed out for clarity but included in calculations. Relative improvement is measured against the base model pretrained on C4.

Table 3. SFT and C4 validation losses for 70M and 160M models comparing default midtraining mixes to continued pretraining on only the midtraining data (100%), averaged across 5 seeds. Bold indicates best performance within each dataset/model combination.

Model	Dataset	Mix	SFT	C4
70M				
Pycode	Pretrain-only	2.656	6.152	
	Starcoder (20%)	2.504	6.032	
	Ctd. pretrain (Starcoder)	2.530	6.109	
GSM8K	Pretrain-only	1.384	6.353	
	Math (12%)	1.339	6.358	
	Ctd. pretrain (Math)	1.383	6.376	
160M				
Pycode	Pretrain-only	2.314	5.254	
	Starcoder (20%)	2.134	5.079	
	Ctd. pretrain (Starcoder)	2.219	5.369	
GSM8K	Pretrain-only	1.163	5.308	
	Math (12%)	1.114	5.230	
	Ctd. pretrain (Math)	1.159	5.326	

6. When and how much midtraining data should be introduced?

Having established that effective midtraining data bridges syntactic patterns in pretraining and posttraining datasets, we now ask a natural question: when should this bridge be introduced, and how much specialized data should be mixed in? Although practitioners routinely tune midtraining mixture weights, the choice of when to begin midtraining, and critically, how start time interacts with mixture weight—has received little systematic study.

We conduct targeted experiments varying both the start point of the midtraining phase (between 12B and 105B tokens into pretraining) and mixture weight (between 10-80% specialized data). We test multiple combinations of starting point and mixture weight to test hypotheses about the interactions between timing and mixture weight, namely: (1) Do timing and mixture weight interact, or do they have independent effects? (2) Can later introduction of specialized data be compensated for by increasing mixture weight? We

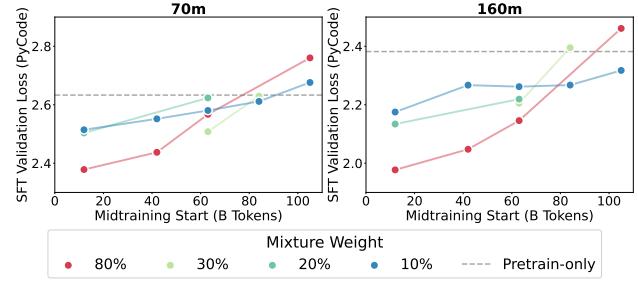


Figure 4. Effect of mixture weight and midtraining phase start on in-domain validation loss for the code mixture. A high mixture weight is beneficial when the midtraining phase begins early, but is detrimental when beginning this phase later.

conduct experiments on the 70m and 160m models with the Starcoder mixture to study these questions, as it is the mix with the strongest in-domain effects.

Timing and mixture weight interact strongly. Figure 4 shows that the optimal mixture weight of specialized data depends critically on when the midtraining phase begins. Early introduction of code with a very high mixture weight (80%) achieves the best in-domain performance. However this relationship reverses later in training, with the high mixture weight (80%) performing substantially worse than the conservative mixture (10%) at 105B tokens.

Compensation through increased mixture fails. Suppose that we have already pretrained a model to a certain number of tokens, and do not want to redo training to accommodate a new midtraining component. Can we make up for this through using higher mixture weights at later start times? When examining the progression of loss values (10% @ 42B, 20% @ 63B, 30% @ 84B), we can see that this is not the case, as shifting from an early introduction point and conservative mixture weight to a late introduction point and aggressive mixture weight degrades performance. This suggests that the model may lack sufficient plasticity to

adapt to a high weight of specialized data late in pretraining.

Relatedly, [Figure 5](#) illustrates how midtraining benefits evolve over the course of pretraining for the 20% Starcoder mix (160m model). We finetune checkpoints from different pretraining steps on Pycode and measure both in-domain and C4 validation loss after fine-tuning. In-domain advantages emerge quickly after midtraining introduction (6k steps), while the C4 retention benefits develop more gradually, becoming apparent after approximately 20k steps. This temporal pattern suggests that early introduction of specialized data provides sufficient time for both immediate domain adaptation and gradual integration with general capabilities.

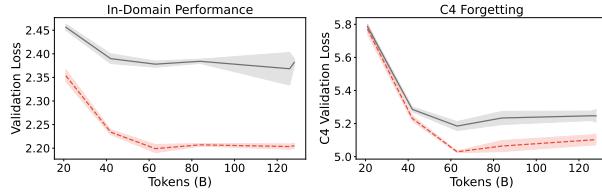


Figure 5. Validation loss and C4 loss for the Starcoder-midtrained model (160M) and base pretrained model *after* supervised finetuning on the Pycode dataset, with each point on the x-axis representing the number of tokens the pretrained checkpoint was trained on.

Finding 4. Midtraining effectiveness depends on a *timing-weight interaction*. For code midtraining, early introduction supports aggressive mixture weights and yields the largest in-domain gains, whereas late introduction makes high mixture weights harmful and favors more conservative mixing

7. How does midtraining change model representations?

Our results suggest that midtraining can improve both in-domain adaptation and pretraining retention when the midtraining mix is well-aligned to the target dataset. We next ask whether this is reflected in the representational changes models undergo during fine-tuning. As an initial probe investigating this, we compare representations between midtrained and base models in the code domain.

We use linear Centered Kernel Alignment (CKA) to measure layer-wise similarity between model states ([Kornblith et al., 2019](#)). We extract activations from all layers using probe datasets (C4 and APPS ([Hendrycks et al., 2021](#))) and compute CKA similarity matrices between four key model states: base pretrained, midtrained (Starcoder), base finetuned, and midtrained finetuned. If midtraining creates better representations for downstream tasks, we expect to

see smaller representational changes during fine-tuning for midtrained models compared to base models.

[Figure 6](#) shows the representational analysis for the 70M model. The midtrained model exhibits greater stability in the final layer after fine-tuning, a pattern consistent across model sizes (see [Appendix G](#) for the remaining results). However, the final fine-tuned models show high similarity regardless of whether models underwent midtraining. These effects are less pronounced for C4, which can be seen in [Appendix H](#). Overall, this is consistent with the view that midtraining acts as a better initiation for downstream SFT, reducing the amount of late-layer change needed to reach a similar fine-tuned representation.

Finding 5. Models exposed to midtraining require smaller representational shifts during fine-tuning, especially in the final layer, indicating smoother adaptation.

8. Related Work

Specific midtrained models Recently, several language model families have adopted midtraining approaches with varying implementation details ([Hu et al., 2024b](#); [Dubey et al., 2024](#); [OLMo Team et al., 2025](#); [Olmo et al., 2025](#); [Chameleon Team, 2024](#)). The midtraining phase duration varies from 2% ([Hu et al., 2024b](#)) to 20% ([Chameleon Team, 2024](#)) of total training, motivating our systematic investigation of timing effects. Common midtraining domains include code, math, instructions, and higher-quality web data ([OLMo Team et al., 2025](#))—the domains we investigate. Beyond general-purpose models, midtraining has shown benefits for specific tasks like RL ([Wang et al., 2025](#)) and GUI agents ([Zhang et al., 2025a](#)). This widespread adoption motivates our questions of when and why midtraining provides downstream benefits.

Staged training and pre-adaptation Several works explore multi-stage pretraining, [Feng et al. \(2024\)](#); [Blakeney et al. \(2024\)](#) focusing on two-stage pretraining and [Zhang et al. \(2025b\)](#) proposing four-stage pretraining. These approaches demonstrate improvements over single-stage pretraining. However, these works evaluate base model performance after pretraining, whereas we focus on the post-finetuning setting to focus on benefits that also affect post-training. Relatedly, domain-adaptive pretraining (DAPT) and related approaches continue pretraining on domain-specific data ([Gururangan et al., 2020](#)). [Krishna et al. \(2023\)](#) show that pretraining on downstream data alone can rival full pretraining when evaluated after fine-tuning, suggesting pretraining-posttraining alignment matters—consistent with our findings. [Mehta et al. \(2023\)](#) find pretraining reduces catastrophic forgetting during sequential fine-tuning; similarly, we observe midtrained models serve as better ini-



Figure 6. CKA analysis of model activations in the 70M model, probed with the APPS code dataset.

tializations with less forgetting. Most similar in spirit to our bridging interpretation is work on pre-finetuning, which selects unlabeled intermediate data to shift a model’s training distribution toward the downstream target (Kang et al., 2024). However, compared to selection-focused approaches, we treat midtraining as a training phase in its own right. Our results show that these schedule choices impact downstream gains and retention, and are based on a similar principle to data selection work.

Concurrent mixing in posttraining A complementary line of work replays pretraining-distribution data during finetuning in order to regularize training, with the goal of mitigating catastrophic forgetting. These include methods that inject selected pretraining samples during finetuning (Liu et al., 2022) as well as rehearsal schemes for multi-stage finetuning (Bai et al., 2024). Scaling analyses have also characterized forgetting as a predictable function of model scale, target data size, and percentage of replay data (Bethune et al., 2025). Although this concept may be similar, replay during fine-tuning intervenes at a different point in the training trajectory: it looks backward, mixing pretraining data as a stability constraint while optimizing the terminal adaptation objective. Midtraining instead looks forward, shaping the initialization so subsequent posttraining is both more effective and less destructive. The two approaches may also be used together.

Stability and Plasticity in training dynamics Recent work addresses stability challenges during continued pretraining. Guo et al. (2024) identify a “stability gap” where performance temporarily drops before recovering when shifting to new domains, Yang et al. (2024) synthesize larger training corpora from small domain-specific datasets, and Lin et al. (2024) introduce selective training on useful tokens only. While these works target training dynamics during continued pretraining, our approach examines how midtraining data selection affects post-fine-tuning performance, representing a complementary focus on end-task effectiveness.

Relationship between Pretraining and Finetuning Several recent works have explored incorporating instruction-formatted data during pretraining. Allen-Zhu & Li (2023) show with an experiment on synthetic Wikipedia-style data that augmenting pretraining data with QA-formatted data improves subsequent fine-tuning, and Jiang et al. (2024) and Cheng et al. (2024) demonstrate this in a practical context as well. Sun & Dredze (2024) find continual pretraining benefits emerge only after fine-tuning, while Springer et al. (2025) show extended pretraining causes catastrophic forgetting (“overtraining”), particularly on math/code domains least aligned with web data. It is possible midtraining may prevent overtraining by introducing specialized data earlier and providing a better initialization for posttraining.

9. Conclusion

We conduct the first systematic investigation of midtraining through controlled experiments. We demonstrate that midtraining benefits are domain-specific, with the most substantial improvements in math and code domains that are not well represented in standard web pretraining corpora. Furthermore, we also find that midtraining mitigates catastrophic forgetting of general language modeling abilities after specific supervised fine-tuning and consistently outperformed continued pretraining on specialized data alone. Furthermore, timing and mixture weight interact, such that the effectiveness of higher mixture weights depends on when specialized data is introduced.

Practically, these results suggest targeting midtraining toward domains whose token patterns differ substantially from base pretraining data, especially when those domains will be used for posttraining. They also motivate exploring timing of data introduction more systematically, and favoring midtraining over continued pretraining. Looking ahead, it will be important to test whether these trends persist at larger scales and across a broader range of domains, and to understand extensions to reinforcement-learning-based posttraining and multi-stage curricula.

Impact Statement

This work studies midtraining as a mechanism for improving the effectiveness of sequential training on different data distributions. We anticipate that by clarifying when and how specialized data should be introduced, our findings may reduce trial-and-error experimentation and thus lower the compute costs required to train and adapt language models to new domains. Our experiments use established datasets and do not introduce new data collection or deployments, however as always, practitioners should continue to follow best practices for data governance and copyright when selecting midtraining data.

References

- Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023. doi: 10.48550/arXiv.2309.14316. URL <https://arxiv.org/abs/2309.14316>.
- Bai, A., Yeh, C.-K., Hsieh, C.-J., and Taly, A. An efficient rehearsal scheme for catastrophic forgetting mitigation during multi-stage fine-tuning. *arXiv preprint arXiv:2402.08096*, 2024. doi: 10.48550/arXiv.2402.08096. URL <https://arxiv.org/abs/2402.08096>. Published in Findings of NAACL 2025 (see arXiv record).
- Beltagy, I., Lo, K., and Cohan, A. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3615–3620, 2020. doi: 10.18653/v1/2020.emnlp-demos.82. URL <https://aclanthology.org/2020.emnlp-demos.82>.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pp. 41–48, 2009. URL <https://doi.org/10.1145/1553374.1553380>.
- Bethune, L., Grangier, D., Busbridge, D., Gualdoni, E., Cuturi, M., and Ablin, P. Scaling laws for forgetting during finetuning with pretraining data injection. *arXiv preprint arXiv:2502.06042*, 2025. doi: 10.48550/arXiv.2502.06042. URL <https://arxiv.org/abs/2502.06042>.
- Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., Skowron, A., Sutawika, L., and Van Der Wal, O. Pythia: A suite for analyzing large language models across training and scaling. In Krause, A., Brunskill, E., Cho, K., Engeldhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/biderman23a.html>.
- Blakenev, C., Paul, M., Larsen, B. W., Owen, S., and Frankle, J. Does your data spark joy? Performance gains from domain upsampling at the end of training, June 2024. URL <http://arxiv.org/abs/2406.03476>. arXiv:2406.03476 [cs].
- Chameleon Team. Chameleon: Mixed-Modal Early-Fusion Foundation Models, May 2024. URL <http://arxiv.org/abs/2405.09818>. arXiv:2405.09818 [cs].
- Cheng, D., Gu, Y., Huang, S., Bi, J., Huang, M., and Wei, F. Instruction pre-training: Language models are supervised multitask learners. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 2529–2550, 2024. doi: 10.18653/v1/2024.emnlp-main.148. URL <https://aclanthology.org/2024.emnlp-main.148>.
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Al-lonsius, D., Song, D., Pintz, D., Livshits, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Nail, G., Mialon, G., Pang, G., Cucurull, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Schiano, I., Kloumann, I., Misra, I., Evtimov, I., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupleti, M.,

- Singh, M., Paluri, M., Kardas, M., Caudy, M., Rodriguez, M., Lithgow-Bertelloni, M., Seastrom, M., White, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keaneally, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Lewis, M., Artetxe, M., Jain, M., Kokkonen, M., Zakharia, M., Peysakhovich, M., Shihadeh, M., Fanton, M., Chen, M., Shabbir, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Howes, R., Rinott, R., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Verma, S., Yamamoto, S., Nie, S., Shiqi, S., Narang, S., Raparth, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Gupta, S., Gupta, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Kohler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Albiero, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wang, X., Wu, X., Wang, X., Xia, X., Wu, X., Gao, X., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Wang, Y., Hao, Y., Qian, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., and Zhao, Z. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Elman, J. L. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993. doi: 10.1016/0010-0277(93)90058-4.
- Feng, S., Prabhumoye, S., Kong, K., Su, D., Patwary, M., Shoeybi, M., and Catanzaro, B. Maximize Your Data’s Potential: Enhancing LLM Accuracy with Two-Phase Pretraining, December 2024. URL <http://arxiv.org/abs/2412.15285>. arXiv:2412.15285 [cs].
- Guo, Y., Fu, J., Zhang, H., Zhao, D., and Shen, Y. Efficient continual pre-training by mitigating the stability gap. *arXiv preprint arXiv:2406.14833*, 2024.
- Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. Don’t stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8342–8360, 2020.
- Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., and Steinhardt, J. Measuring coding challenge competence with apps. In *Advances in Neural Information Processing Systems (NeurIPS) 2021*, 2021. URL <https://proceedings.neurips.cc/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper.pdf>.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J., Vinyals, O., and Sifre, L. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20122–20134, 2022. URL <https://arxiv.org/abs/2203.15556>.
- Hu, S., Bai, Z., Chen, T., Jiang, S., Jiang, X., Liu, C., Li, W., Lai, G., Cui, Z., Zhang, C., Zhao, R., Han, J., Dong, S., Fang, J., Shi, B., Wang, T., Liu, Z., Tang, Z., Li, L., Wu, B., Wang, T., Zheng, C., Zhao, T., Zhang, Z., Fu, W., Dai, B., Zhou, J., Li, R., Zheng, Z., Xu, H., Sun, X., Zhou, B., Jiao, S., Li, J., Cao, B., Zhao, X., Lu, Y., Qi, Z., Shi, J., Xiang, H., Wu, J., and Sun, M. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024a.
- Hu, S., Tu, Y., Han, X., He, C., Cui, G., Long, X., Zheng, Z., Fang, Y., Huang, Y., Zhao, W., Zhang, X., Thai, Z. L., Zhang, K., Wang, C., Yao, Y., Zhao, C., Zhou, J., Cai, J., Zhai, Z., Ding, N., Jia, C., Zeng, G., Li, D., Liu, Z., and Sun, M. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies, April 2024b. URL <http://arxiv.org/abs/2404.06395>. arXiv:2404.06395 [cs].
- Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. Codeseachnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.
- Jiang, Z., Sun, Z., Shi, W., Rodriguez, P., Zhou, C., Neubig, G., Lin, X., Yih, W.-t., and Iyer, S. Instruction-tuned language models are better knowledge learners. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5421–5434, 2024. doi: 10.18653/v1/2024.acl-long.296. URL <https://aclanthology.org/2024.acl-long.296>.
- Kang, F., Just, H. A., Sun, Y., Jahagirdar, H., Zhang, Y., Du, R., Sahu, A. K., and Jia, R. Get more for less: Principled data selection for warming up fine-tuning in llms. *arXiv preprint arXiv:2405.02774*, 2024.
- Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C. Measuring catastrophic forgetting in neural

- networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 2019. URL <https://proceedings.mlr.press/v97/kornblith19a.html>.
- Krishna, K., Garg, S., Bigham, J., and Lipton, Z. Downstream datasets make surprisingly good pretraining corpora. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12207–12222, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.682. URL <https://aclanthology.org/2023.acl-long.682/>.
- Li, H., Ding, L., Fang, M., and Tao, D. Revisiting catastrophic forgetting in large language model tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4297–4308, Miami, Florida, USA, 2024a. Association for Computational Linguistics. URL <https://aclanthology.org/2024.findings-emnlp.249/>.
- Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., Garg, S., Xin, R., Muennighoff, N., Heckel, R., Mercat, J., Chen, M., Gururangan, S., Wortsman, M., Albalak, A., Bitton, Y., Nezhurina, M., Abbas, A., Hsieh, C.-Y., Ghosh, D., Gardner, J., Kilian, M., Zhang, H., Shao, R., Pratt, S., Sanyal, S., Ilharco, G., Daras, G., Marathe, K., Gokaslan, A., Zhang, J., Chandu, K., Nguyen, T., Vasiljevic, I., Recht, B., Zettlemoyer, L., Iyer, S., Zhuang, T., Liang, P., Rush, A., Jain, N., Raunak, V., Cardie, C., and Schmidt, L. Datacomp-lm: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*, 2024b.
- Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shli-azhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Rao, N., Stojnic, R., Allamanis, M., Laffitte, P., Rustamov, P. K., Valter, K., Mittal, C., Tesfamikael, K. K., Murati, C., Lee, S., Wan, A. Q., Suharyanto, A., Copet, J., So, D. R., Kolubako, L., Pina, G. M., Behtash, S., Moskovskiy, D., Siddarth, D., Luce-Rainville, N., Dehghani, M., Szafraniec, M., Cardozo, P., Jitsev, J., Kochmar, E., Torralba, A., Radev, D., Rush, A. M., Nakov, P., Wang, T., Zuo, W., Echikson, H., Schuelke, L., Carmichael, J., Sadagopan, K. S., Ling, Z., Kwiatkowski, C., Lohn, A., Mueller, J., and Floetenmeyer, H. d. V. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- Lightning AI. Litgpt. <https://github.com/Lightning-AI/litgpt>, 2023.
- Lin, Z., Gou, Z., Gong, Y., Liu, X., Shen, Y., Xu, R., Lin, C., Yang, Y., Jiao, J., Duan, N., et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.
- Liu, Z., Xu, Y., Xu, Y., Qian, Q., Li, H., Ji, X., Chan, A., and Jin, R. Improved fine-tuning by better leveraging pre-training data. *arXiv preprint arXiv:2111.12292*, 2022. doi: 10.48550/arXiv.2111.12292. URL <https://arxiv.org/abs/2111.12292>.
- Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.
- Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., and Zhang, Y. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2024.
- Mehta, S. V., Patil, D., Chandar, S., and Strubell, E. An empirical investigation of the role of pre-training in lifelong learning. *Journal of Machine Learning Research*, 24(214):1–50, 2023. URL <http://jmlr.org/papers/v24/22-0496.html>.
- Olmo, T., Ettinger, A., Bertsch, A., Kuehl, B., Graham, D., Heineman, D., Groeneveld, D., Brahman, F., Timbers, F., Ivison, H., Morrison, J., Poznanski, J., Lo, K., Soldaini, L., Jordan, M., Chen, M., Noukhovitch, M., Lambert, N., Walsh, P., Dasigi, P., Berry, R., Malik, S., Shah, S., Geng, S., Arora, S., Gupta, S., Anderson, T., Xiao, T., Murray, T., Romero, T., Graf, V., Asai, A., Bhagia, A., Wettig, A., Liu, A., Rangapur, A., Anastasiades, C., Huang, C., Schwenk, D., Trivedi, H., Magnusson, I., Lochner, J., Liu, J., Miranda, L. J. V., Sap, M., Morgan, M., Schmitz, M., Guerquin, M., Wilson, M., Huff, R., Bras, R. L., Xin, R., Shao, R., Skjonsberg, S., Shen, S. Z., Li, S. S., Wilde, T., Pyatkin, V., Merrill, W., Chang, Y., Gu, Y., Zeng, Z., Sabharwal, A., Zettlemoyer, L., Koh, P. W., Farhadi, A., Smith, N. A., and Hajishirzi, H. Olmo 3, 2025. URL <https://arxiv.org/abs/2512.13961>.
- OLMo Team, Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., Lambert, N., Schwenk, D., Tafjord, O., Anderson, T., Atkinson, D., Brahman, F., Clark, C., Dasigi, P., Dziri,

- N., Guerquin, M., Ivison, H., Koh, P. W., Liu, J., Malik, S., Merrill, W., Miranda, L. J. V., Morrison, J., Murray, T., Nam, C., Pyatkin, V., Rangapur, A., Schmitz, M., Skjonsberg, S., Wadden, D., Wilhelm, C., Wilson, M., Zettlemoyer, L., Farhadi, A., Smith, N. A., and Hajishirzi, H. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2025.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL <https://arxiv.org/abs/1910.10683>.
- Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. Curriculum learning: A survey, 2022. URL <https://arxiv.org/abs/2101.10382>.
- Springer, J. M., Goyal, S., Wen, K., Kumar, T., Yue, X., Malladi, S., Neubig, G., and Raghunathan, A. Over-trained language models are harder to fine-tune. *arXiv preprint arXiv:2503.19206*, 2025. URL <https://arxiv.org/abs/2503.19206>.
- Sun, K. and Dredze, M. Amuro and char: Analyzing the relationship between pre-training and fine-tuning of large language models. *arXiv preprint arXiv:2408.06663*, 2024. doi: 10.48550/arXiv.2408.06663. URL <https://arxiv.org/abs/2408.06663>.
- Toshniwal, S., Moshkov, I., Narendhiran, S., Gitman, D., Jia, F., and Gitman, I. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv:2402.10176*, 2024.
- Wang, Z., Zhou, F., Li, X., and Liu, P. Octothinker: Mid-training incentivizes reinforcement learning scaling, 2025.
- Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.
- Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL <https://aclanthology.org/W17-4413/>.
- Yang, Z., Band, N., Li, S., Candès, E., and Hashimoto, T. Synthetic continued pretraining. *arXiv preprint arXiv:2409.07431*, 2024.
- Yue, X., Qu, X., Zhang, G., Fu, Y., Huang, W., Sun, H., Su, Y., and Chen, W. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.
- Zhang, J., Ding, Z., Ma, C., Chen, Z., Sun, Q., Lan, Z., and He, J. Breaking the data barrier – building gui agents through task generalization, 2025a.
- Zhang, X., Duan, F., Xu, L., Zhou, Y., Wang, S., Weng, R., Wang, J., and Cai, X. Frame: Boosting llms with a four-quadrant multi-stage pretraining strategy, 2025b. URL <https://arxiv.org/abs/2502.05551>.
- Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., Ghosh, A., Xiao, L., Ettinger, A., Chang, S., Peng, B., Dai, Y., Jain, H., Cruz, S., Gupta, A., Zhang, H., Srinivasan, S., Berg-Kirkpatrick, T., Hovy, E., Manning, C. D., Zettlemoyer, L., and Levy, O. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*, 2023.

A. Theoretical Analysis

Here, we present a simple theoretical analysis on the influence of midtraining on forgetting of the original data distribution, as well as on in-domain loss. We use a minimal set of assumptions commonly used in first-order optimization analyses in order to obtain a tractable bound. However, we do not claim that these assumptions hold globally for large language models.

Let the population loss on the pretraining distribution be $J_P(\theta)$ and the population loss on the SFT distribution T be $J_T(\theta)$. Let θ_0 be the parameters of a model immediately before finetuning, and let θ_K represent the SFT loss after K steps of gradient descent on the SFT dataset.

A.1. Assumptions and Notation

Assumption A.1 (Local smoothness). J_P is L_P -smooth and J_T is L_T -smooth on a neighborhood containing the iterates $\{\theta_k\}_{k=0}^K$, and on line segments between them i.e., $\|\nabla J(\theta) - \nabla J(\theta')\| \leq L\|\theta - \theta'\|$ for $J \in \{J_P, J_T\}$.

Assumption A.2 (Step size). The posttraining step size satisfies $\eta \leq 1/L_T$.

A.2. Standard inequalities

We use two standard consequences of L -smoothness:

Lemma A.3 (Quadratic upper bound / descent lemma). *If f is L -smooth on a convex domain, then for all x, y in the domain,*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2. \quad (7)$$

Lemma A.4 (GD decrease lemma). *If f is L -smooth and $\eta \leq 1/L$, then for the GD update $\theta^+ = \theta - \eta \nabla f(\theta)$,*

$$f(\theta^+) \leq f(\theta) - \frac{\eta}{2}\|\nabla f(\theta)\|^2. \quad (8)$$

Assume that the model undergoes a midtraining stage at some point before finetuning, where $\theta_0(t, w)$ represents the model after midtraining, with start timestep t and mixture weight w . Additionally, define θ_t^{pre} as the parameters immediately before midtraining, and let $\delta(t, w) = \theta_0(t, w) - \theta_t^{\text{pre}}$ be the change in parameters during the midtraining phase.

We are interested in the **forgetting** of the model after K steps of finetuning, namely:

$$\Delta_P(K) := J_P(\theta_K) - J_P(\theta_0) \quad (9)$$

A.3. Bounding forgetting over K steps

We start by bounding a one-step change in J_P in the direction $\theta_{t+1} = \theta_t - \eta \nabla J_T(\theta_t)$. Because J_P is L_P smooth, we can apply the usual Equation 7:

$$\begin{aligned} J_P(\theta_{t+1}) &\leq J_P(\theta_t) - \eta \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle + \frac{L_P \eta^2}{2} \|\nabla J_T(\theta_t)\|^2 \\ J_P(\theta_{t+1}) - J_P(\theta_t) &\leq -\eta \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle + \frac{L_P \eta^2}{2} \|\nabla J_T(\theta_t)\|^2. \end{aligned} \quad (10)$$

We can sum the one-step inequality over $t = 0, \dots, K-1$ to yield a telescoping sum:

$$\begin{aligned} \sum_{t=0}^{K-1} (J_P(\theta_{t+1}) - J_P(\theta_t)) &\leq -\eta \sum_{t=0}^{K-1} \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle + \frac{L_P \eta^2}{2} \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2 \\ \Delta_P(K) = J_P(\theta_K) - J_P(\theta_0) &\leq \underbrace{-\eta \sum_{t=0}^{K-1} \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle}_{\text{gradient alignment}} + \underbrace{\frac{L_P \eta^2}{2} \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2}_{\text{energy term}}. \end{aligned} \quad (11)$$

We can see that there is one ‘‘gradient alignment’’ based term and one ‘‘energy’’ based term. We can also bound the ‘‘energy’’ term with [Equation 8](#).

$$\begin{aligned} J_t(\theta_{t+1}) &\leq J_T(\theta_t) - \frac{\eta}{2} \|\nabla J_t(\theta_t)\|^2 \\ \eta \|\nabla J_t(\theta_t)\|^2 &\leq 2(J_t(\theta_t) - J_t(\theta_{t+1})) \end{aligned}$$

Again using a telescoping sum, Summing over $t = 0, \dots, K - 1$ yields

$$\eta \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2 \leq 2(J_T(\theta_0) - J_T(\theta_K))$$

Let J_T^* represent the best possible loss on T . Then we can write

$$\begin{aligned} \eta \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2 &\leq 2(J_T(\theta_0) - J_T^*) \\ \eta^2 \sum_{t=0}^{K-1} \|\nabla J_T(\theta_t)\|^2 &\leq 2\eta(J_T(\theta_0) - J_T^*) \end{aligned}$$

Substituting back into [Equation 11](#), we get the final bound:

$$\Delta_P(K) \leq -\eta \sum_{t=0}^{K-1} \langle \nabla J_P(\theta_t), \nabla J_T(\theta_t) \rangle + L_P \eta (J_T(\theta_0) - J_T^*) \quad (12)$$

We now show how midtraining can potentially impact forgetting through initialization.

Lemma A.5 (Initialization effect on the energy term). *Assume J_T is L_T -smooth on a convex neighborhood containing θ and $\theta + \delta$. Then for any displacement δ ,*

$$J_T(\theta + \delta) \leq J_T(\theta) + \langle \nabla J_T(\theta), \delta \rangle + \frac{L_T}{2} \|\delta\|^2. \quad (13)$$

In particular, taking $\theta = \theta_t^{\text{pre}}$ and $\delta = \delta(t, w) := \theta_0(t, w) - \theta_t^{\text{pre}}$ yields

$$J_T(\theta_0(t, w)) \leq J_T(\theta_t^{\text{pre}}) + \langle \nabla J_T(\theta_t^{\text{pre}}), \delta(t, w) \rangle + \frac{L_T}{2} \|\delta(t, w)\|^2. \quad (14)$$

Consequently, a sufficient condition for midtraining to decrease the SFT loss at initialization, $J_T(\theta_0(t, w)) \leq J_T(\theta_t^{\text{pre}})$, is

$$\langle \nabla J_T(\theta_t^{\text{pre}}), \delta(t, w) \rangle \leq -\frac{L_T}{2} \|\delta(t, w)\|^2. \quad (15)$$

Proof. This is a direct application of the descent lemma ([Theorem A.3](#)) with $f = J_T$, $x = \theta$, and $y = \theta + \delta$:

$$\begin{aligned} J_T(\theta + \delta) &\leq J_T(\theta) + \langle \nabla J_T(\theta), (\theta + \delta) - \theta \rangle + \frac{L_T}{2} \|(\theta + \delta) - \theta\|^2 \\ &= J_T(\theta) + \langle \nabla J_T(\theta), \delta \rangle + \frac{L_T}{2} \|\delta\|^2. \end{aligned}$$

Substituting $\theta = \theta_t^{\text{pre}}$ and $\delta = \delta(t, w)$ gives [Equation 14](#). Finally, $J_T(\theta_0(t, w)) \leq J_T(\theta_t^{\text{pre}})$ holds whenever the sum of the linear and quadratic terms in [Equation 14](#) is nonpositive, i.e., whenever [Equation 15](#) holds. \square

Lemma A.6 (Bounding the midtraining displacement). *Suppose midtraining runs for m steps starting from θ_t^{pre} and produces $\theta_0(t, w)$. Let $\{\varphi_u\}_{u=0}^m$ be the midtraining iterates with*

$$\varphi_0 = \theta_t^{\text{pre}}, \quad \varphi_m = \theta_0(t, w), \quad \delta(t, w) = \varphi_m - \varphi_0.$$

Assume the midtraining updates have the form

$$\varphi_{u+1} = \varphi_u - \alpha s(t+u) g_u, \quad u = 0, \dots, m-1, \quad (16)$$

where $\alpha > 0$ is the midtraining step size, $s(\cdot) \in [0, 1]$ is nonincreasing (representing a loss of plasticity or other proxy for diminishing leverage of later updates compared to earlier ones), and g_u is the (stochastic) gradient used at step u . Define the cumulative plasticity over the block

$$S(t) := \sum_{u=0}^{m-1} s(t+u).$$

If $\|g_u\| \leq G(w)$ for all u in the block, then

$$\|\delta(t, w)\| \leq \alpha G(w) S(t), \quad (17)$$

and hence

$$\frac{L_T}{2} \|\delta(t, w)\|^2 \leq \frac{L_T}{2} \alpha^2 G(w)^2 S(t)^2. \quad (18)$$

Proof. Summing the increments in Equation 16 yields

$$\delta(t, w) = \varphi_m - \varphi_0 = \sum_{u=0}^{m-1} (\varphi_{u+1} - \varphi_u) = -\alpha \sum_{u=0}^{m-1} s(t+u) g_u.$$

Taking norms and applying the triangle inequality,

$$\begin{aligned} \|\delta(t, w)\| &= \left\| -\alpha \sum_{u=0}^{m-1} s(t+u) g_u \right\| \leq \alpha \sum_{u=0}^{m-1} s(t+u) \|g_u\| \\ &\leq \alpha G(w) \sum_{u=0}^{m-1} s(t+u) = \alpha G(w) S(t), \end{aligned}$$

which proves Equation 17. Squaring and multiplying by $L_T/2$ gives Equation 18. \square

Plugging into the bound. Applying Equation 12 with $\theta_0 = \theta_0(t, w)$ gives

$$\Delta_P(K) \leq -\eta \sum_{k=0}^{K-1} \langle \nabla J_P(\theta_k), \nabla J_T(\theta_k) \rangle + L_P \eta \left(J_T(\theta_0(t, w)) - J_T^* \right)$$

By Lemma A.5 (descent lemma applied to J_T at θ_t^{pre} with displacement $\delta(t, w)$),

$$J_T(\theta_0(t, w)) \leq J_T(\theta_t^{\text{pre}}) + \langle \nabla J_T(\theta_t^{\text{pre}}), \delta(t, w) \rangle + \frac{L_T}{2} \|\delta(t, w)\|^2$$

By Lemma A.6, $\|\delta(t, w)\| \leq \alpha G(w) S(t)$, hence

$$\frac{L_T}{2} \|\delta(t, w)\|^2 \leq \frac{L_T}{2} \alpha^2 G(w)^2 S(t)^2$$

Combining these inequalities yields

$$\begin{aligned} \Delta_P(K) &\leq -\eta \sum_{k=0}^{K-1} \langle \nabla J_P(\theta_k), \nabla J_T(\theta_k) \rangle \\ &\quad + L_P \eta \left[(J_T(\theta_t^{\text{pre}}) - J_T^*) + \langle \nabla J_T(\theta_t^{\text{pre}}), \delta(t, w) \rangle + \frac{L_T}{2} \alpha^2 G(w)^2 S(t)^2 \right]. \end{aligned} \quad (19)$$

B. Pretraining Settings

We pretrained models from scratch on the C4 dataset. All three models were trained for 128B tokens or approximately 61k steps, with very similar settings (documented in [Table 4](#)). L40S GPUs were used for all pretraining and midtraining runs. Models were trained with the LitGPT library ([Lightning AI, 2023](#)).

Table 4. Core pretraining hyperparameters for Pythia-70M, 160M, 410M, and 1B.

Hyperparameter	70M	160M	410M	1B
Global batch size	1024	1024	1024	1024
Micro batch size	16	16	8	4
LR schedule	Cosine w/ 10% warmup			
Max LR	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Min LR	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
Optimizer	AdamW	AdamW	AdamW	AdamW
Betas	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)
Weight decay	0.1	0.1	0.1	0.1
Precision	BF16	BF16	BF16	BF16
Num ranks	4	4	8	8

C. Posttraining Settings

We fine-tuned all models on four downstream datasets: Pycode (our 5K-sample subset of CodeSearchNet-Python), GSM8K (7.5K math problems), LIMA (1K instruction examples), and SciQ (13.7K science questions). For GSM8K only, the prompt/question portion was masked during loss; for the others the loss was computed over the full sequence. A summary of the datasets is given in [Table 5](#). All runs used a cosine learning rate schedule with 10% linear warmup, trained for 4 epochs, global batch size 64, and micro-batch size 16 for 70M/160M (8 for 410M). Peak learning rates were selected by grid search on the base pretrained checkpoint before midtraining, and the LR grid is given in [Table 6](#). Selected LRs for the final checkpoint of each model size are given in [Table 7](#).

Table 5. Finetuning datasets.

Dataset	# Train Samples	Prompt masked
Pycode (CodeSearchNet-Python subset)	5,000	No
GSM8K	7,500	Yes
LIMA	1,000	No
SciQ	13,679	Yes

Table 6. Grid of candidate peak learning rates swept during tuning.

LR grid

4e-6, 8e-6, 1e-5, 2e-5, 4e-5, 5e-5, 6e-5, 7e-5, 8e-5, 9e-5, 1e-4, 1.2e-4, 1.4e-4, 1.6e-4, 1.8e-4, 2e-4, 2.4e-4, 4e-4, 5e-4, 6e-4, 8e-4, 1e-3, 2e-3, 3e-3, 4e-3, 6e-3
--

Table 7. Selected peak learning rates for fine-tuning (cosine schedule with 10% warmup).

Dataset	70M	160M	410M	1B
GSM8K	8e-4	4e-4	4e-4	7e-05
LIMA	1.2e-4	5e-5	5e-5	2e-05
Pycode	1e-3	5e-4	4e-4	9e-05
SciQ	8e-4	2.4e-4	6e-4	9e-05

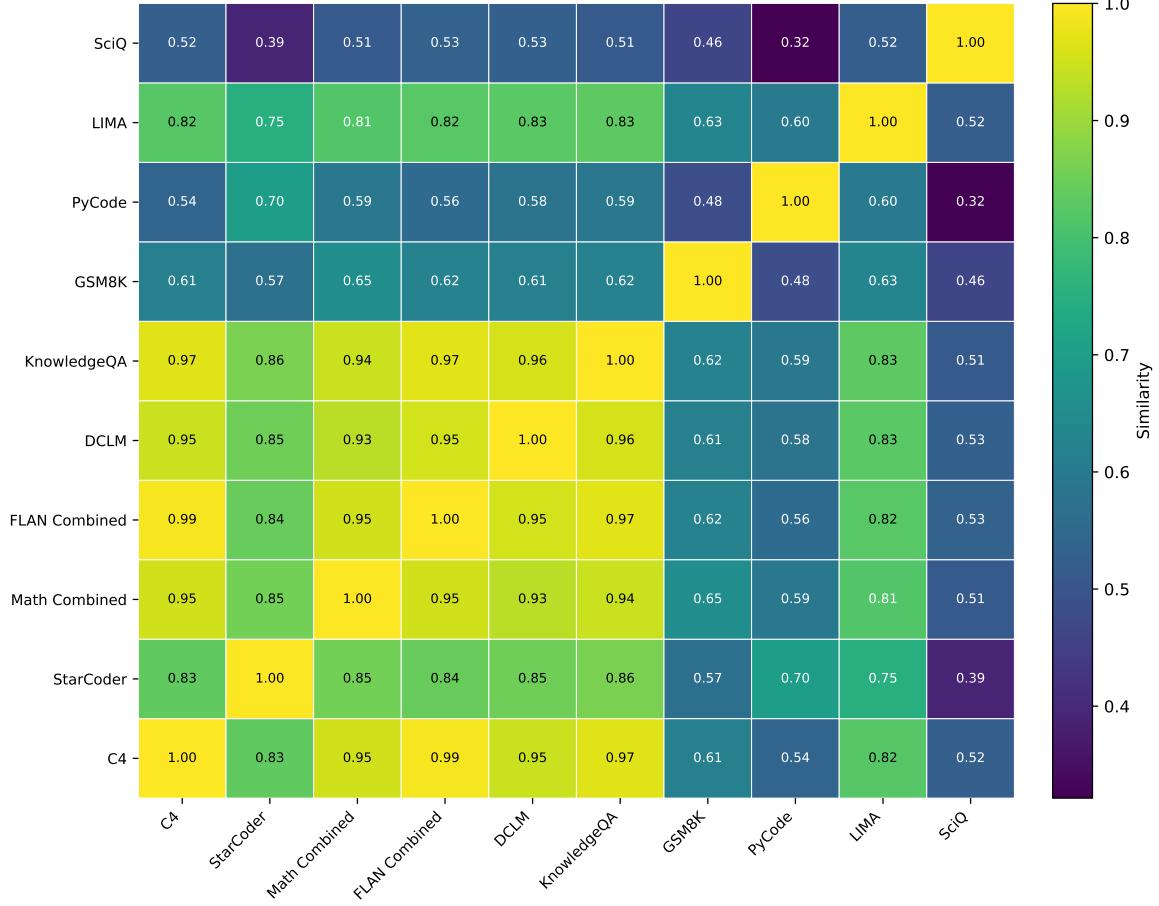
D. Dataset Similarity Matrix

We compute dataset similarity using surface-level token statistics after initial experimentation with embedding models gave implausible results for code datasets' similarities to other natural language datasets. For each pair of pretrain/midtrain

and downstream datasets, we sample $\max(\text{dataset_size}, 10,000)$ examples. Midtrain mixes are simulated by their actual compositions (e.g., Starcoder is treated as 20% Starcoder + 80% C4). From the (possibly mixed) texts we build unigram frequency vectors at a token level, normalize to probabilities, and compute: vocabulary Jaccard, overlap ratio, token-frequency cosine similarity, and a Jensen–Shannon-based similarity. These are combined as

$$\text{Combined} = 0.4 \cdot \text{cosine} + 0.3 \cdot \text{Jaccard} + 0.3 \cdot \text{JS_similarity},$$

and used to fill the similarity matrix (diagonal entries are 1). This mixture-aware score reflects both specialty content and dilution by C4. [Figure 7](#) shows the resulting similarity matrix between pre/midtrain datasets and SFT datasets.



[Figure 7](#). Token-based similarity matrix for pre/midtrain and SFT datasets. Note that these midtrain datasets are corrected for mix weight in this matrix.

E. SFT in-domain loss and C4 Losses after Finetuning for 70m and 160m models

[Table 8](#) depicts validation losses as well as C4 validation losses after finetuning on each SFT dataset, for the 70m and 160m models.

Midtraining Bridges Pretraining and Posttraining Distributions

Table 8. SFT and C4 validation losses for 70M, 160M, and 410M models across downstream datasets and midtraining mixtures, averaged across 5 seeds for each SFT dataset. Bold values indicate best performance within each dataset and model size combination.

Model Size	Downstream Dataset	Midtrain Mix	SFT Val Loss	C4 Val Loss
70m	Pycode	C4	2.656	6.152
		Starcoder (20%)	2.504	6.032
		Math (12%)	2.603	6.116
		FLAN (5%)	2.802	6.400
		KnowledgeQA (20%)	2.628	6.117
		DCLM (20%)	2.584	6.052
160m	GSM8K	C4	1.384	6.353
		Starcoder (20%)	1.353	6.317
		Math (12%)	1.339	6.358
		FLAN (5%)	1.368	6.352
		KnowledgeQA (20%)	1.367	6.376
		DCLM (20%)	1.368	6.352
410m	LIMA	C4	4.333	4.124
		Starcoder (20%)	4.346	4.136
		Math (12%)	4.362	4.146
		FLAN (5%)	4.342	4.110
		KnowledgeQA (20%)	4.290	4.110
		DCLM (20%)	4.324	4.097
70m	SciQ	C4	3.159	7.703
		Starcoder (20%)	3.187	7.804
		Math (12%)	3.187	7.971
		FLAN (5%)	3.161	7.888
		KnowledgeQA (20%)	3.142	7.567
		DCLM (20%)	3.147	7.703
160m	Pycode	C4	2.314	5.254
		Starcoder (20%)	2.134	5.079
		Math (12%)	2.332	5.277
		FLAN (5%)	2.318	5.257
		KnowledgeQA (20%)	2.306	5.232
		DCLM (20%)	2.305	5.215
160m	GSM8K	C4	1.163	5.308
		Starcoder (20%)	1.134	5.315
		Math (12%)	1.114	5.230
		FLAN (5%)	1.152	5.299
		KnowledgeQA (20%)	1.145	5.287
		DCLM (20%)	1.149	5.303
410m	LIMA	C4	3.828	3.578
		Starcoder (20%)	3.810	3.581
		Math (12%)	3.795	3.569
		FLAN (5%)	3.836	3.559
		KnowledgeQA (20%)	3.736	3.560
		DCLM (20%)	3.792	3.549
410m	SciQ	C4	2.705	4.423
		Starcoder (20%)	2.728	4.474
		Math (12%)	2.740	4.343
		FLAN (5%)	2.708	4.427
		KnowledgeQA (20%)	2.673	4.159
		DCLM (20%)	2.671	4.377
410m	Pycode	C4	2.151	5.032
		Starcoder (20%)	1.971	4.608
		Math (12%)	2.159	5.109
		FLAN (5%)	2.152	4.920
		KnowledgeQA (20%)	2.151	5.052
		DCLM (20%)	2.159	5.109
410m	GSM8K	C4	1.043	4.952
		Starcoder (20%)	1.029	4.923
		Math (12%)	1.004	4.872
		FLAN (5%)	1.050	5.089
		KnowledgeQA (20%)	1.043	4.928
		DCLM (20%)	1.056	5.052
410m	LIMA	C4	3.446	3.178
		Starcoder (20%)	3.403	3.162
		Math (12%)	3.452	3.180
		FLAN (5%)	3.471	3.175
		KnowledgeQA (20%)	3.468	3.173
		DCLM (20%)	3.463	3.170

Continued on next page

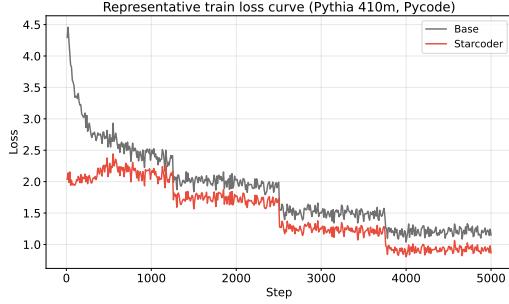


Figure 8. Representative training loss curve for a midtrained model and base model on Pycode, for Pythia-410m. The midtrained model starts with a lower training loss, and maintains a slight gap throughout training.

Model Size	Downstream Dataset	Midtrain Mix	SFT Val Loss	C4 Val Loss
SciQ	C4		2.247	3.646
	Starcoder (20%)		2.223	3.593
	Math (12%)		2.255	3.610
	FLAN (5%)		2.233	3.581
	KnowledgeQA (20%)		2.226	3.541
	DCLM (20%)		2.240	3.647

F. Representative training loss curves for midtrained vs. base models

Figure 8 shows a representative training loss curve for a midtrained model when its domain is aligned to SFT data.

G. Additional CKA results on APPS

Figure 9 and Figure 10 display the CKA layer similarity for 160m and 410m models.

H. CKA results on C4

Figure 11, Figure 12, and Figure 13 show the CKA layer similarity for all model sizes with C4 as a probe.

I. Statement on LLM Usage

Large language models (LLMs) were used to assist with refining writing in this submission, including summarizing paragraphs in order to shorten the submission, correcting grammar, and giving suggestions to improve organization. LLMs were not used in the ideation process and analyses and experimental setups were designed fully by the authors. Copilot and other coding agents were used to generate some utility scripts in the process of coding.

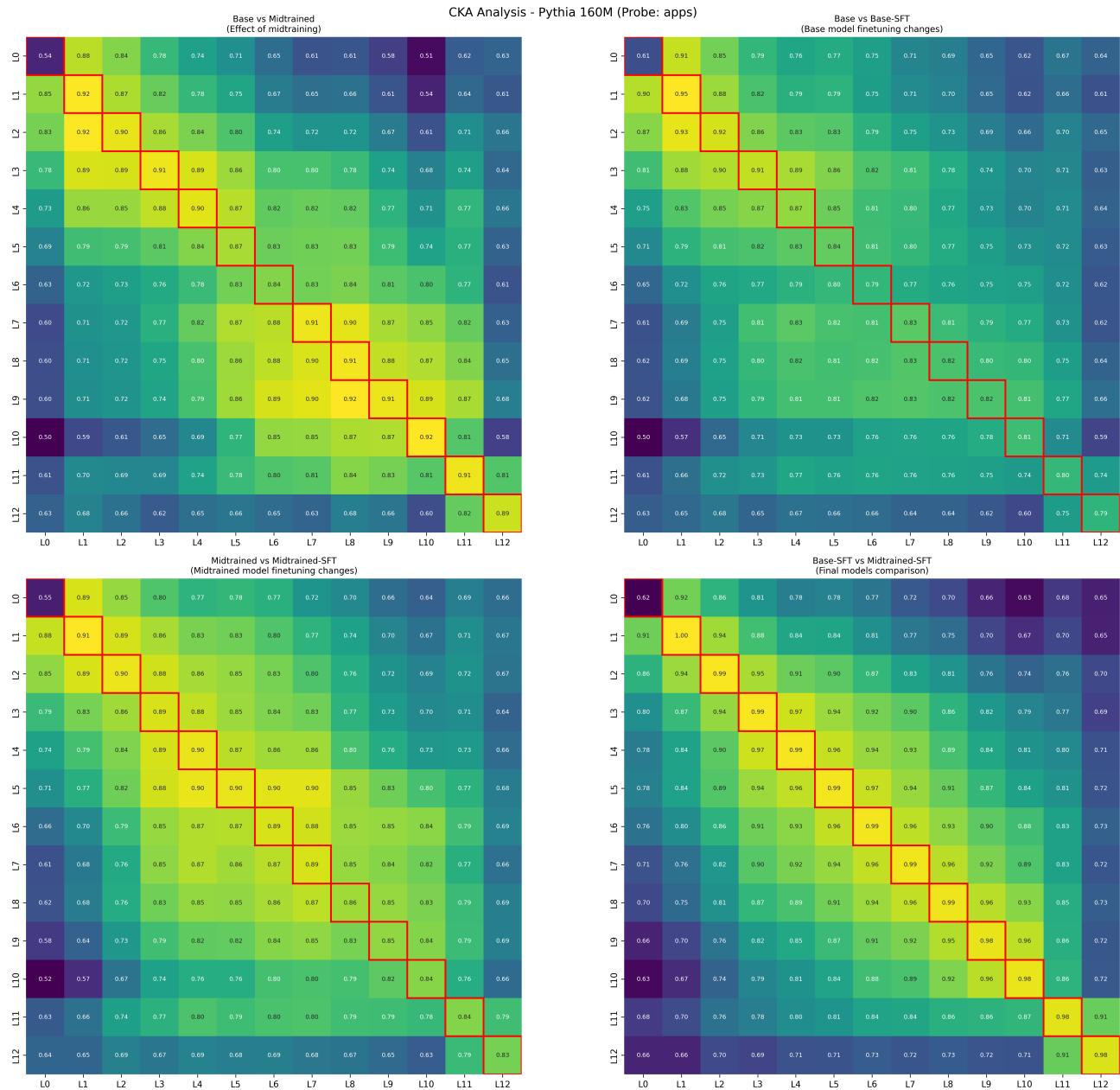


Figure 9. CKA layer analysis for Pythia-160M with APPS as a probe.

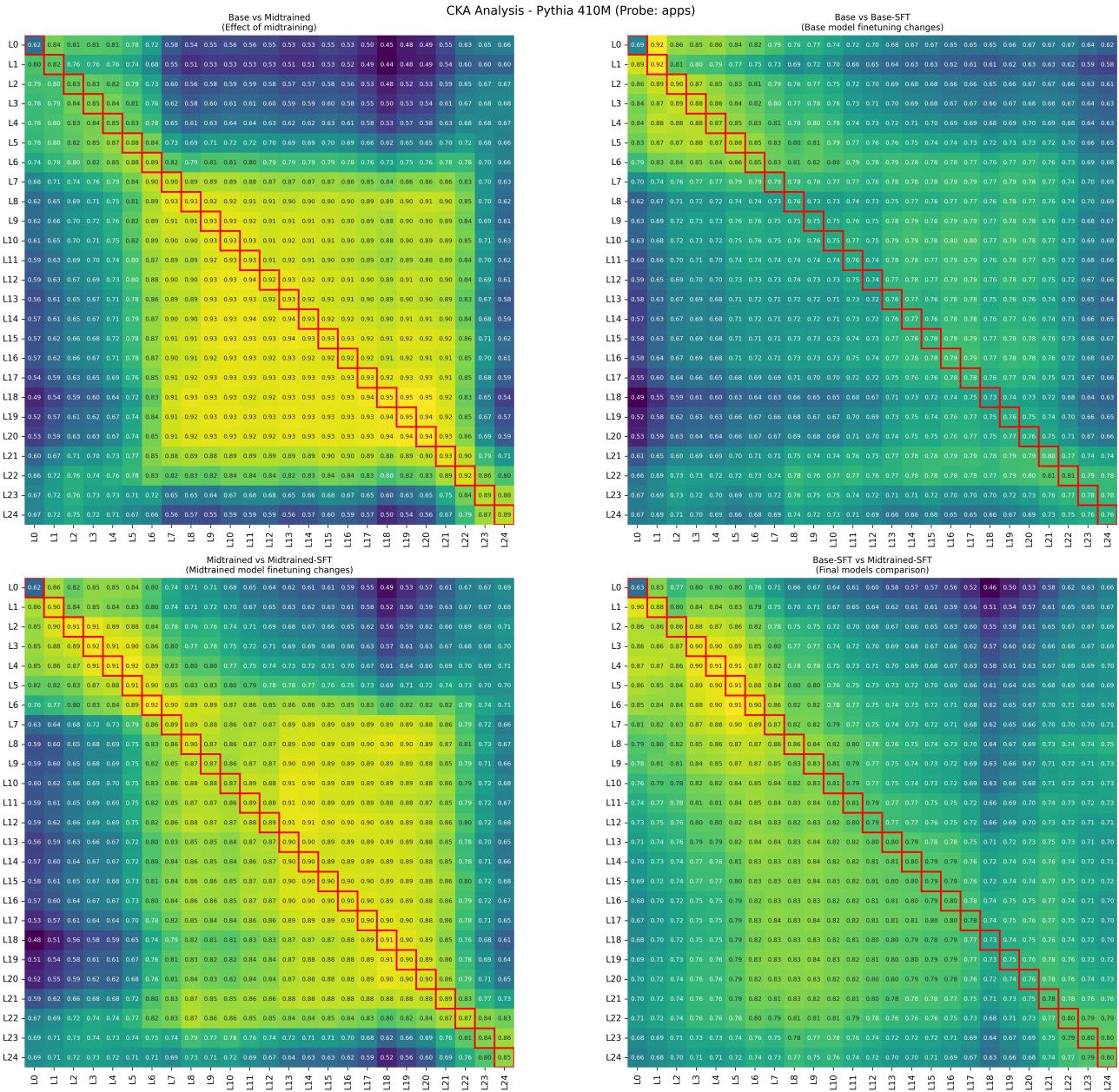


Figure 10. CKA layer analysis for Pythia-410M with APPS as a probe.

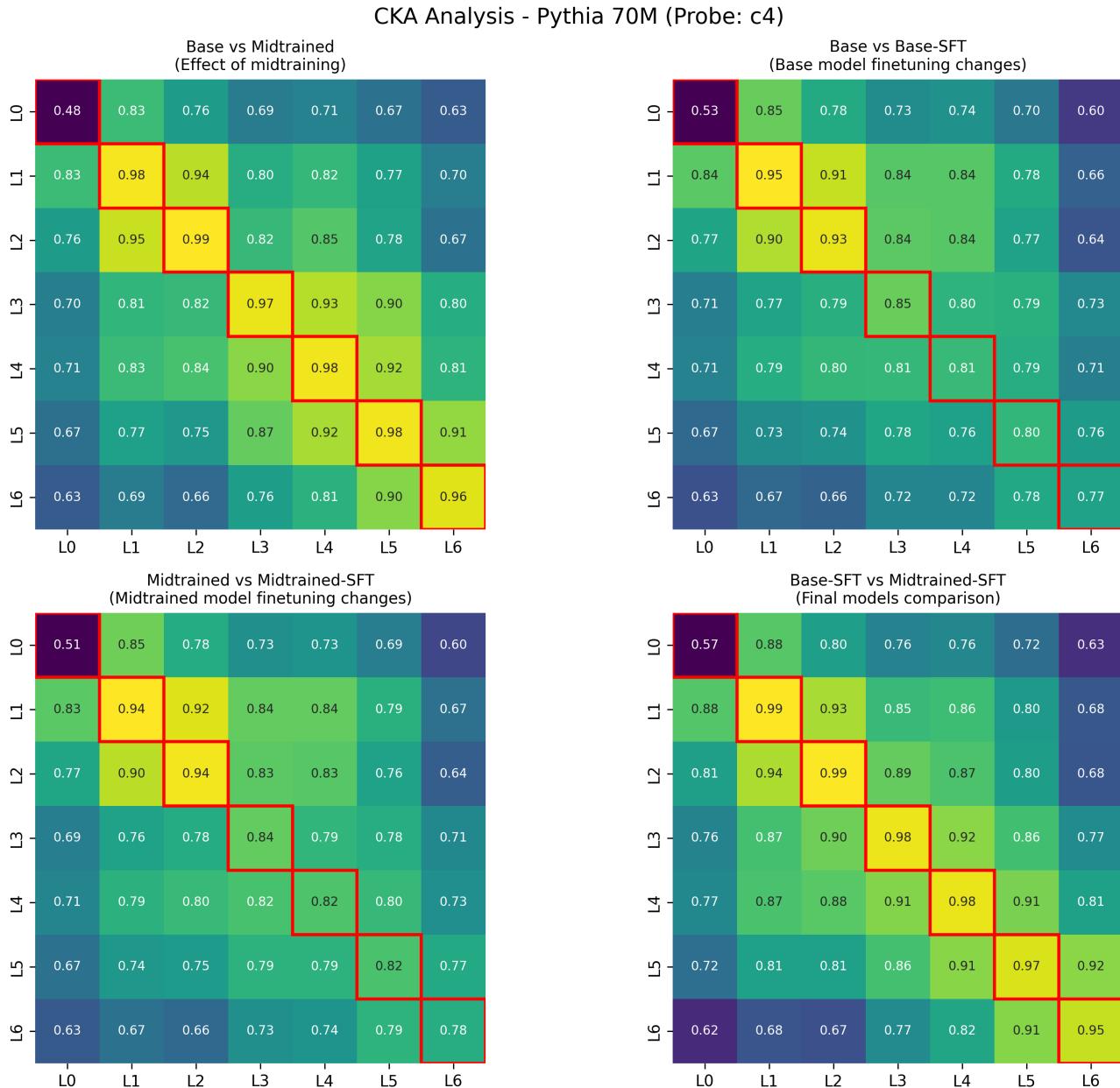


Figure 11. CKA layer analysis for Pythia-70M with C4 as a probe.

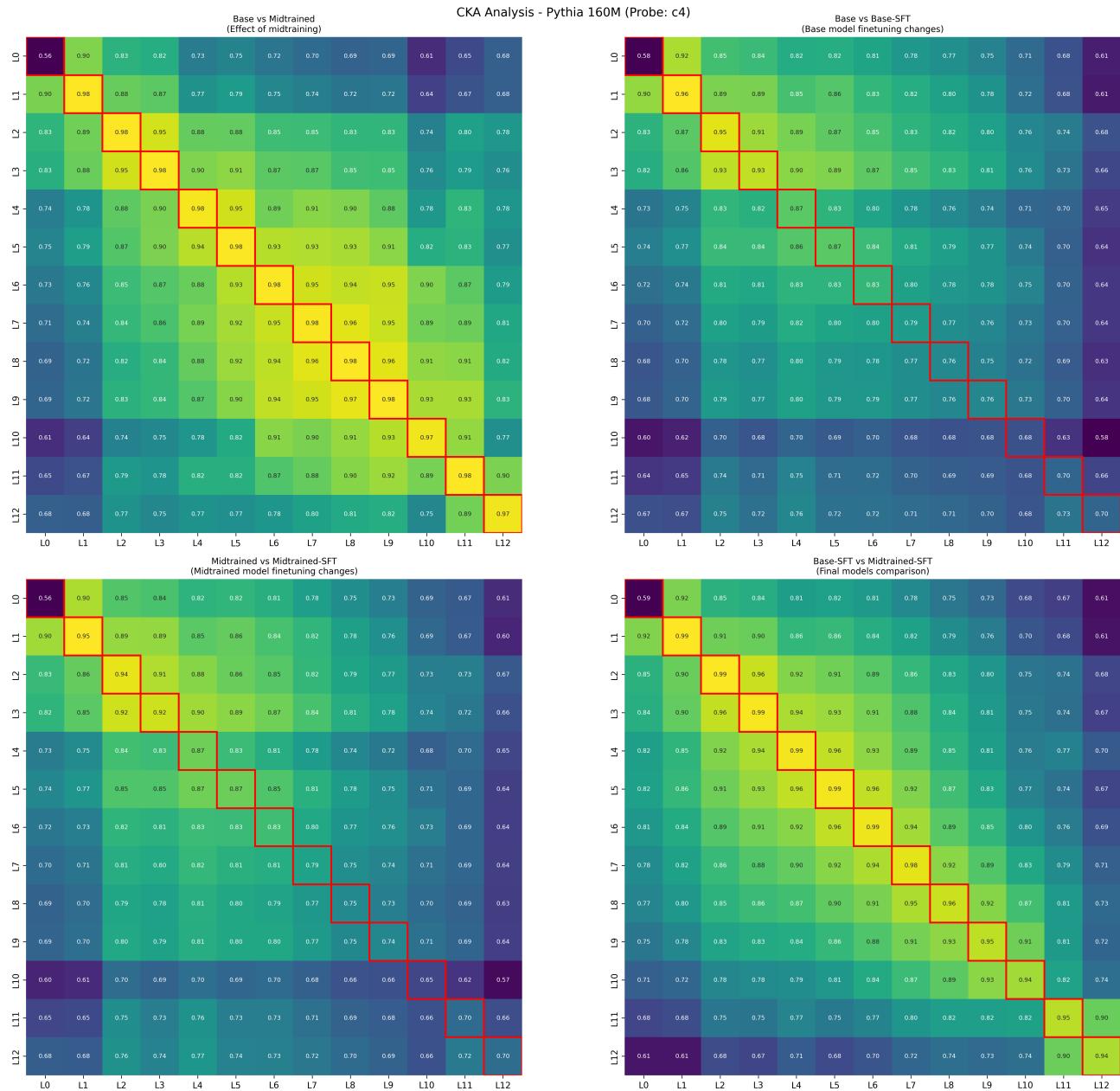


Figure 12. CKA layer analysis for Pythia-160M with C4 as a probe.

Midtraining Bridges Pretraining and Posttraining Distributions

