Efficient Model Development through Recycling Fine-tuning

Anonymous ACL submission

Abstract

Modern LLMs struggle with efficient updates, as each new pretrained version requires repeating expensive alignment processes. This challenge also applies to domain- or language-specific models, where fine-tuning on specialized data must be redone for every new base model release. We propose a method to recycle fine-tuning across model versions by transferring weight changes—or *diff vectors*—from a previously fine-tuned model to a new base model. We empirically validate this approach across different open-weight model versions, showing that transferred diff vectors can significantly enhance the performance of the new base model, often achieving results competitive with direct fine-tuning. Through controlled experiments, we establish that fine-tuning transfer is most effective when the source and target models are linearly connected in the parameter space. Additionally, we apply our approach to multilingual model development and show that recycling fine-tuning can improve performance on target-language tasks without additional training. Furthermore, we demonstrate that recycling fine-tuning provides a stronger and more computationally efficient starting point for fine-tuning. Finally, we introduce an iterative *recycling-then-finetuning* approach for continuous model development, which further enhances efficiency and effectiveness. Our findings suggest that recycling fine-tuning is a viable strategy for reducing training costs while maintaining model performance.

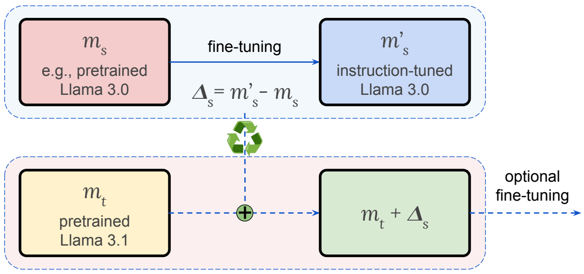

Figure 1: To recycle fine-tuning (e.g., instruction tuning) from a *source* model version s (e.g., Llama 3.0) to a *target* version t (Llama 3.1), we compute the diff vector $\Delta_s = m'_s - m_s$, where m'_s is the fine-tuned model (instruction-tuned Llama 3.0) and m_s is the base model (pretrained Llama 3.0) at version s. We then add Δ_s to the new base model (pretrained Llama 3.1) to approximate the fine-tuned model at version t (instruction-tuned Llama 3.1).

1 Introduction

Modern LLMs are developed in two main stages: (1) *pretraining* on vast text corpora using self-supervised objectives such as next-word prediction, and (2) *post-training*, which involves additional alignment steps, such as supervised fine-tuning and reinforcement learning to align the model with human preferences. While this approach yields powerful and versatile LLMs, it also poses significant challenges for model updates. Specifically, each new version of the pretrained model requires repeating the alignment process, which is costly, especially for frequent updates. This issue is compounded when developing domain- or language-specific models, as fine-tuning on specialized data must be redone for every new base model release.

In this paper, we explore methods to improve LLM training efficiency by effectively recycling fine-tuning across model versions. More specifically, we propose reusing and incorporating updates (i.e., *weight changes*) from a previous model version to enhance the training efficiency and performance on the newer version. Such an approach reduces the need to retrain from scratch, thereby cutting down training time and computational resources.

Our approach (see Figure 1) explores the *diff* vector $\Delta_s = m'_s - m_s$, which represents the difference between a fine-tuned model m'_s (e.g., instruction-tuned) and its base model m_s (pretrained). Intuitively, Δ_s encodes the specific changes in model parameters during fine-tuning, and can be used to transfer knowledge from a *source* model version s (e.g., Llama 3) to a *target* (*newer*) model version t (Llama 3.1). We hypothesize that given the same fine-tuning data, these fine-tuned models exhibit linear relationships between versions: $m'_s - m_s \approx m'_t - m_t$. This allows us to approximate a fine-tuned version of a newer model without additional fine-tuning: $m'_t \approx m_t + \Delta_s$. This is similar to the concept of *task vectors* (Ilharco et al., 2023), but rather than transfer between different tasks using the same model, we focus on transfer between different versions of the model trained on the same data.

We first evaluate the feasibility of our proposed approach by transferring diff vectors across different versions of open-weight models, including Llama (Dubey et al., 2024), OLMo (OLMo et al., 2024), and Tülu (Lambert et al., 2024b). Our results demonstrate that fine-tuning can be effectively recycled across model versions. Specifically, merging the diff vector Δ_s from an older version s to the base model m_t of a newer version t results in a model $m_t + \Delta_s$ that significantly improves m_t's performance on various tasks, often achieving competitive results with its fine-tuned counterpart (m'_t) without requiring additional fine-tuning.

To shed light on when fine-tuning transfer is most effective, we conduct controlled experiments using intermediate checkpoints of OLMo 2 as different model versions. By fine-tuning these models on the same dataset and transferring Δ_s across checkpoints, we observe that our recycling is most successful when the source and target models reside in a linearly connected region of the parameter space, indicating linear mode connectivity.

Building on these insights, we further conduct a case study on multilingual model development. We fine-tuned the instruction-tuned Llama 3 for specific languages, then transferring the resulting diff vectors to the instruction-tuned Llama 3.1. Recycling fine-tuning produces models that outperform the instruction-tuned Llama 3.1 in the target language without requiring additional training, further demonstrating its effectiveness.

Additionally, we investigate whether the merged model $m_t + \Delta_s$ serves as a computationally efficient and effective starting point for fine-tuning. Our experiments show that initializing fine-tuning from this merged model accelerates convergence and improves accuracy compared to training m_t from scratch. This suggests that recycling fine-tuning can be a beneficial intermediate step in scenarios where retraining is feasible.

Finally, we explore a continuous model development scenario where new versions are released regularly. We introduce an iterative *recycling-then-finetuning* approach that incrementally accumulates fine-tuning updates from previous versions. Our experiments show that this method consistently enhances both training efficiency and model performance. In summary, our main contributions are:

- Introducing a method for recycling fine-tuning across model versions via diff vector transfer.
- Demonstrating that this approach reduces training costs while maintaining competitive performance.
- Establishing conditions under which fine-tuning transfer is most effective, particularly in linearly connected model spaces.
- Validating the method on multilingual models, showing improved language-specific performance without retraining.
- Proposing recycling-then-finetuning strategies for efficient model development, further enhancing efficiency and performance.

2 Recycling fine-tuning across model versions

In the development of today's LLMs, when a new and improved pretrained model is released, fine-tuned models (such as those optimized for specific tasks or languages) need to be retrained to take advantage of the improvements. To address this challenge, we explore transferring weight changes from a source model version s to a target model version t, denoted as $\mathcal{R}_{s \to t}$, to minimize or eliminate the need for retraining. We first evaluate the feasibility of this approach by directly merging (adding) the diff vector $\Delta_s = m'_s - m_s$, which captures parameter adaptations from the base model m_s to its fine-tuned counterpart m'_s in version s, onto the new base model m_t in version t, without additional gradient-based training. Our results show that fine-tuning can be effectively recycled across model versions, as $m_t + \Delta_s$ often performs comparably to its fine-tuned counterpart m'_t.

2.1 Experiment setup

We experiment with different open-weight models, including Llama (Dubey et al., 2024),

Model	GSM8K	MATH	ARC_C	GPQA	MMLU	IFEval
Llama 3.0 8B Instruct	81.1	28.8	82.4	**31.5**	64.9	**76.6**
Llama 3.0 8B	55.6	17.3	79.7	22.3	66.7	34.5
+ $\Delta_{3.1}$	**82.8**	**44.7**	**83.0**	25.9	**70.0**	**76.6**
Llama 3.1 8B Instruct	**86.5**	**50.3**	**83.8**	31.3	**72.9**	80.5
Llama 3.1 8B	56.6	19.3	79.2	21.9	66.8	36.4
+ $\Delta_{3.0}$	79.8	29.9	82.9	**32.6**	65.1	**83.3**

Table 1: Recycling fine-tuning significantly improves the new base model's performance across various tasks, reaching levels comparable to retraining. $\Delta_{3.0}$ and $\Delta_{3.1}$ represent the diff vectors between Llama Instruct and Llama for versions 3.0 and 3.1, respectively.

OLMo (OLMo et al., 2024), and Tülu (Lambert et al., 2024b). We emphasize that our goal is not to achieve state-of-the-art results but instead to access the feasibility of recycling fine-tuning (by transferring weight changes) between model versions. As such, we explore both transfer directions: from an *older* version to a *newer* version and vice versa. We provide additional results for OLMo and Tülu in Appendix 7.

We evaluate the merged model $m_t + \Delta_s$ on a diverse set of benchmarks, including general knowledge with MMLU (Hendrycks et al., 2021a), math with GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021c), reasoning with ARC_C and GPQA (Rein et al., 2024), and instruction-following with IFEval (Zhou et al., 2023). We compare its performance against fine-tuning m_t directly (i.e., m'_t).

2.2 Results and discussion

Recycling fine-tuning substantially boosts the new base model's performance: Table 1 shows our results when recycling fine-tuning (i.e., instruction tuning) between Llama 3.0 and Llama 3.1. Strikingly, adding the diff vector Δ_s from a different model version can transform a non-instruction-tuned model (i.e., pretrained Llama 3.0 and Llama 3.1) into one (Llama 3.0 + $\Delta_{3.1}$ and Llama 3.1 + $\Delta_{3.0}$, respectively) that can follow instructions well. For instance, this approach yields +42.1% and +46.9% absolute accuracy improvements on the instruction-following IFEval benchmark over Llama 3.0 and Llama 3.1, respectively. Large gains are also observed across the board on math and reasoning benchmarks, including +27.2% over Llama 3.0 and +23.2% over Llama 3.1 on GSM8K. Overall, Llama 3.0 benefits more from this recycling fine-tuning approach than Llama 3.1. The absolute accuracy improvements via $\Delta_{3.1}$ are consistently higher than those of Llama 3.1 + $\Delta_{3.0}$, suggesting that advanced knowledge and instruction-following abilities can be efficiently transferred to another version of the model without extensive retraining.

Recycling fine-tuning can achieve performance comparable to retraining: Our results also demonstrate that the merged model $m_t + \Delta_s$ can perform on par with its fine-tuned counterpart m'_s on different tasks. This is particularly true for Llama 3.0 + $\Delta_{3.1}$, which matches or surpasses Llama 3.0 Instruct on five out of six tasks we consider. Interestingly, Llama 3.1 + $\Delta_{3.0}$ outperforms LLama 3.1 Instruct on the GPQA and IFEval benchmarks. This is a testament to the notion that the diff vector can effectively encode advanced reasoning and instruction-following capabilities. Overall, our results suggest that recycling fine-tuning provides an effective and extremely low-cost method to improve model performance when retraining is prohibitively expensive. However, the conditions under which this approach is effective remain murky, which motivates us to conduct controlled experiments where we fine-tune various model versions on the same data and evaluate the impact of recycling fine-tuning across versions (see §3.2).

3 When is recycling fine-tuning effective?

Having demonstrated the effectiveness of recycling fine-tuning across model versions, we now conduct controlled experiments to better understand when this approach is most effective. At a high level, we treat different checkpoints of a pretrained model as distinct model versions. We then fine-tune these model versions on the same data and evaluate the impact of reusing fine-tuning across them. Our results show that fine-tuning transfer is most effective when the source and target models are close within a linearly connected region of the parameter space,

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	13.2	19.4	24.4	64.5	65.5
+ Δ_1		**26.6**	32.0	27.5	19.6
+ Δ_2	**19.0**		**39.8**	25.9	17.3
+ Δ_3	14.3	25.0		68.6	70.3
+ Δ_4	11.8	18.0	22.6		**77.1**
+ Δ_5	11.9	16.0	24.0	**72.9**	
Direct FT	45.1	50.7	60.4	75.7	75.5

Table 2: GSM8K accuracies indicating that stronger models are better at utilizing recycled fine-tuning, which is most effective when models are close in the parameter space. $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2, while Δ_i denotes the diff vector from the fine-tuning of version i. See Appendix C for MATH500 results.

indicating linear mode connectivity.

3.1 Experiment setup

We experiment with OLMo 2 7B's publicly available intermediate pretrained checkpoints.[1] Base OLMo 2 were trained in two stages: (1) general web-based pretraining stage (stage 1) and (2) mid-training (stage 2) using high-quality web data and domain-specific data to enhance STEM-related capabilities. We select five checkpoints: $\mathcal{M}_1$ (early-stage 1, at 300K steps), $\mathcal{M}_2$ (mid-stage 1, 600K steps), $\mathcal{M}_3$ (end-stage 1, 929K steps), $\mathcal{M}_4$ (mid-stage 2, 6K steps), and $\mathcal{M}_5$ (end-stage 2, 12K steps). We treat $\mathcal{M}_i$ as distinct model versions and investigate recycling fine-tuning between them in both directions: from earlier to later versions and vice versa. The former can minimize or eliminate the need for retraining, aligning with our recycling goal, while the latter can be beneficial in specific scenarios (e.g., when incorporating new fine-tuning into an earlier base model yields better results for a particular use case).

Due to our limited computational resources, supervised fine-tuning with a large instruction tuning dataset would be prohibitively expensive. As such, we fine-tune all model versions using Tülu 3's math reasoning instruction tuning data subset, which includes Tülu 3 Persona MATH, GSM, and Algebra (220K examples total). Unless stated otherwise, we fine-tune each model for 30K steps with a learning rate of 5e-8 and a batch size of 8 on 4 NVIDIA A100-80G GPUs.[2]

We evaluate our models on GSM8K and the MATH500 (Hendrycks et al., 2021c) subset of MATH, which includes competition-level math problems of varying difficulty. We choose these datasets because fine-tuning on Tülu 3's math reasoning data significantly improves performance on them, allowing for a clearer analysis of the impact of recycling fine-tuning across model versions.

3.2 Results and discussion

Stronger models are better at leveraging recycled fine-tuning: While recycling fine-tuning can improve performance for $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, the merged models still lag far behind their fine-tuned counterparts. On GSM8K, the accuracy gaps between the best merged models and fine-tuned versions are 26.2%, 24.2%, 20.6% for $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, respectively. In contrast, for $\mathcal{M}_4$, this gap narrows to 2.8%. Notably, recycling fine-tuning from $\mathcal{M}_4$ to $\mathcal{M}_5$ surpasses direct fine-tuning (77.1% vs. 75.6%). Similar trends are observed on MATH500. This pattern suggests an emergent ability—effective use of recycled fine-tuning only emerges when the target base model is sufficiently strong. In other words, the benefits of recycling only become significant beyond a certain capability level.

Recycling works best when models are close in the parameter space: Our results also suggest that recycling is most effective when the source and target models are closely connected in the parameter space. On both GSM8K and MATH 500, models $\mathcal{M}_1$ and $\mathcal{M}_2$ benefit more from Δ_3 than from Δ_4 or Δ_5 (Δ_i denotes the diff vector from version $\mathcal{M}_i$). Similarly, $\mathcal{M}_4$ and $\mathcal{M}_5$ gain more from Δ_3 than from Δ_1 or Δ_2. Overall, $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ form a mutually beneficial group, as do $\mathcal{M}_4$ and $\mathcal{M}_5$. However, recycling fine-tuning across these two groups can degrade performance. Specifically, $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ do not benefit from Δ_4 and Δ_5, while $\mathcal{M}_4$ and $\mathcal{M}_5$ typically benefit only from Δ_3.[3]

[1] https://huggingface.co/allenai/OLMo-2-1124-7B

[2] We use the AdamW optimizer with a linear scheduler and a warmup ratio of 0.03. Following OLMo 2 and Tülu 3, we disable dropout and exclude weight decay for embeddings. The sequence length is 2048. We use AI2's Open-Instruct (Lambert et al., 2024b) and OLMES (Gu et al., 2024) repositories for training and evaluation, respectively.

[3] The only exception is $\mathcal{M}_4$ benefiting from $\mathcal{M}_1$ and $\mathcal{M}_2$ on MATH500.

4 Efficient multilingual model development

Having explored recycling fine-tuning for developing task-specific models, we now turn toward applying this approach to multilingual model development. Specifically, we aim to recycle multilingual fine-tuning across different versions of the base model. Unlike previous experiments, we fine-tune instruction-tuned models instead of pretrained ones using language-specific data. This follows today's common practice of starting with instruction-tuned models for multilingual tasks. A challenge in this setting is that state-of-the-art LLMs often include multilingual data in pretraining and instruction tuning, making it unclear if language-specific fine-tuning is still necessary. An interesting question is how effective our recycling fine-tuning approach is when applied on top of a multilingual instruction-tuned model. Our results demonstrate that recycling fine-tuning remains effective in this scenario, provided the base model is still outperformed by its previous monolingual counterpart. In such a scenario, our method creates models that outperform the instruction-tuned Llama 3.1 on target-language tasks without further fine-tuning.

4.1 Experiment setup

We separately fine-tune Llama 3.0 Instruct (m_s) on instruction tuning data for three languages: Malagasy, Sinhala, and Turkish. For monolingual instruction tuning, we use Cohere For AI's Aya dataset (Singh et al., 2024b) for Malagasy (14.6K examples) and Sinhala (14.5K examples), and InstrucTurca (Altinok, 2024) for Turkish (16.7K examples).[4] The training follows the procedure in §3.2. After training, we compute the diff vector $\Delta_s = m'_s - m_s$ and add it to Llama 3.1 Instruct m_t. Here, we focus on a low-resource setting and refrain from additional training on language-specific data. We evaluate the models using the Global MMLU benchmark (Singh et al., 2024a), comparing the merged model $m_t + \Delta_s$ with the base model m_t.

4.2 Results and discussion

Recycling fine-tuning is effective for multilingual model development: Our results in Table 3 highlight the advantages of recycling fine-tuning for multilingual model development. For Malagasy and Turkish, applying the difference vector from Llama 3.0 Instruct to Llama 3.1 yields substantial accuracy gains (+4.7% and +15.5% respective accuracy improvements). Additionally, it improves the fine-tuned version of Llama 3.0 Instruct on Malagasy (+1.5%) while maintaining comparable performance on Turkish. This is particularly appealing to the multilingual community, as it enables model improvement at an extremely low cost by leveraging prior fine-tuning and an updated base model.

[4]The original InstrucTurca dataset contains 2.58M examples, but we sampled 6.5% of the data (roughly 16.7K examples) to simulate a low-resource setting.

Model	Malagasy	Sinhala	Turkish
Llama 3.0 8B Instruct	23.1	23.3	30.8
+ FT	30.8	29.0	43.2
Llama 3.1 8B Instruct	27.6	**33.0**	27.7
+ $\Delta_{3.0}$	**32.3**	32.3	**43.2**

Table 3: Recycling fine-tuning boosts multilingual performance on Global MMLU without retraining. $\Delta_{3.0}$ represents the diff vector between Llama 3.0 Instruct and its fine-tuned (FT) version.

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	13.2	19.4	24.4	64.5	65.5
+ $\Delta_1 \rightarrow$ FT		$56.9_{+30.3}$	$62.8_{+30.8}$	$77.8_{+50.3}$	$78.6_{+59.0}$
+ $\Delta_2 \rightarrow$ FT	$\mathbf{50.1}_{+31.1}$		$62.7_{+22.9}$	$\mathbf{78.6}_{+52.7}$	$78.7_{+61.4}$
+ $\Delta_3 \rightarrow$ FT	$48.5_{+34.2}$	$\mathbf{57.6}_{+32.6}$		$77.6_{+9.0}$	$\mathbf{78.8}_{+8.5}$
+ $\Delta_4 \rightarrow$ FT	$48.2_{+36.4}$	$56.7_{+38.7}$	$\mathbf{63.7}_{+41.1}$		$77.2_{+0.1}$
+ $\Delta_5 \rightarrow$ FT	$48.1_{+36.2}$	$55.6_{+39.6}$	$63.5_{+39.5}$	$76.2_{+3.3}$	
Direct FT	45.1	50.7	60.4	75.7	75.5

Table 4: GSM8K accuracies showing that recycling fine-tuning provides a stronger starting point for fine-tuning (FT). Numbers in subscript indicate improvement over the baseline without fine-tuning. $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2, while Δ_i denotes the diff vector from the fine-tuning of version i. See Appendix D for MATH500 results.

On the other hand, for Sinhala, recycling fine-tuning provides no advantage since Llama 3.1 Instruct already performs better than the previously fine-tuned Llama 3.0 Instruct. That said, even in this case, recycling does not significantly affect performance.

5 Recycling as a starting point for fine-tuning

So far, we have considered a scenario where fine-tuning is reused across model versions without additional gradient-based training. Now, we switch gears to investigate whether the merged model $m_t + \Delta_s$ can serve as a stronger and more com-

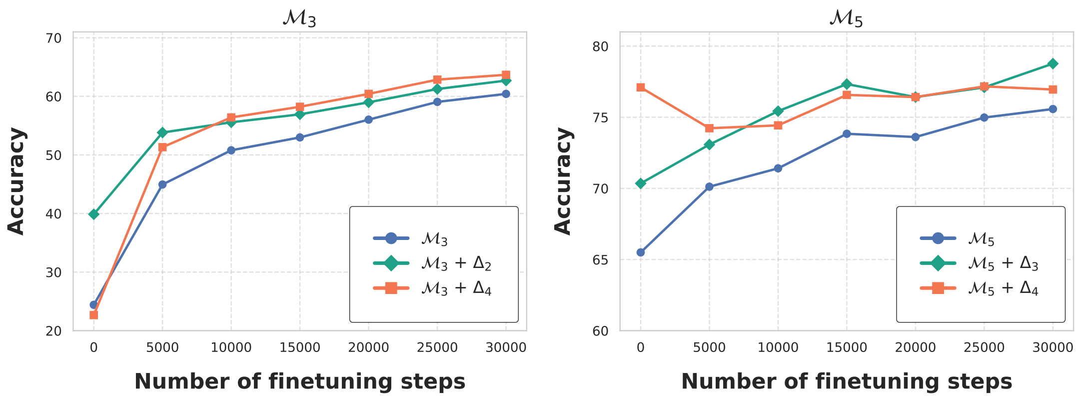

Figure 2: GSM8K performance showing that recycling fine-tuning provides a more computationally efficient starting point for fine-tuning. $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2, while Δ_i denotes the diff vector from the fine-tuning of version i. Appendix E contains results for other model versions.

putationally efficient starting checkpoint for fine-tuning. In our controlled experiments, we compare fine-tuning the merged model $m_t + \Delta_s$ with directly fine-tuning m_t. Our results demonstrate that initializing fine-tuning with $m_t + \Delta_s$ often leads to faster convergence and higher performance on both seen and unseen tasks. This indicates that recycling fine-tuning between model versions can be a useful intermediate step in scenarios where retraining is feasible.

5.1 Experiment setup

For training, we follow the procedure outlined in §3.2. For evaluation, we use GSM8K and MATH500, as well as several additional datasets to assess how well our *recycling-then-finetuning* approach generalizes to unseen tasks, including PhD-level science questions with $\text{GPQA}_{\text{Diamond}}$ (Rein et al., 2024), tabular math word problems with TabMWP (Lu et al., 2023), and elementary school math word problems with ASDiv (Miao et al., 2020).

5.2 Results and discussion

Recycling-then-finetuning can substantially boost performance: Our results are summarized in Tables 4, 5 and Figure 2. As can be seen, recycling-then-finetuning offers significant improvements over the vanilla recycling approach (without additional training) on both GSM8K and MATH500. On GSM8K, the largest accuracy improvements are +36.4%, +39.6%, +41.1%, +52.7%, and +61.4% for $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$, and $\mathcal{M}_5$, respectively. These gains are most notable for weaker base models ($\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$) regardless of the diff vector used or for stronger base models paired with a weak diff vector (e.g., $\mathcal{M}_5 + \Delta_1$). Interestingly, for each base model $\mathcal{M}_i$, fine-tuning helps bridge the performance gap between the merged models $\mathcal{M}_i + \Delta_j$ ($i \neq j$). For example, fine-tuning significantly boosts the performance of $\mathcal{M}_5 + \Delta_1$ and $\mathcal{M}_5 + \Delta_2$ from 10.6% and 17.3% to 78.6% and 78.7%, respectively, closing the gap with the fine-tuned versions of $\mathcal{M}_5 + \Delta_3$ (78.8%) and $\mathcal{M}_5 + \Delta_4$ (77.2%). This reduces the need to pre-select the best diff vector when multiple options are available. Notably, recycling-then-finetuning generally outperforms standard fine-tuning (without recycling) regardless of the diff vector used.

Recycling-then-finetuning can offer faster convergence: Using the merged model $m_t + \Delta_s$ as the initial checkpoint enhances training efficiency. As illustrated in Figure 2, on the GSM8K dataset, $m_t + \Delta_s$ not only converges significantly faster than m_t during fine-tuning but also achieves higher peak accuracy. These results demonstrate the advantages of recycling-based fine-tuning over standard fine-tuning without recycling. Overall, our findings suggest that recycling-then-fine-tuning offers a cost-effective way to reduce the number of fine-tuning steps, thereby improving training efficiency.

Effect of recycling-then-finetuning on model generalization Recycling-then-finetuning does not negatively impact model generalization. As shown in Table 5, this method attains strong zero-shot generalization on the three unseen tasks, similar to standard fine-tuning without recycling. These results suggest that recycling-then-finetuning does not lead to overfitting, which demonstrates its

	GPQA$_{\text{Diamond}}$	TabMWP	ASDiv
$\mathcal{M}_5$ (OLMo 2 7B)	25.2	22.4	28.1
+ Δ_4	**28.2**	46.4	**82.1**
Direct FT	26.2	**48.5**	81.8

Table 5: Recycling-then-finetuning does not hinder model generalization. Here, we apply the diff vector Δ_4 from a previous OLMo 2 pretrained version ($\mathcal{M}_4$) to a newer pretrained version ($\mathcal{M}_5$).

broad applicability across diverse tasks.

6 Iterative recycling-then-finetuning for improved performance and efficiency

We now leverage the insights from our previous experiments to explore a continuous model development setting where new versions of a pretrained model are regularly released. The basic idea behind our approach is an *iterative recycling-then-finetuning* strategy that incrementally incorporates fine-tuning updates, i.e., diff vectors, from past model versions. Instead of applying only the latest diff vector to the new base model, we recycle all previous diff vectors. Specifically, inspired by the success of our recycling-then-finetuning strategy, the diff vector at the current model version is carried forward to the next for subsequent fine-tuning. Our experiments show that this iterative recycling approach consistently improves both training efficiency and model performance.

6.1 Iterative recycling-then-finetuning

We treat the five intermediate checkpoints of OLMo 2 7B—$\mathcal{M}_1$, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$, $\mathcal{M}_5$ (described in §3.2) as different model versions of the pretrained OLMo 2 model. Our iterative recycling-then-finetuning algorithm, outlined in Algorithm 1, works as follows: At each iteration i, we first apply the most recent diff vector, Δ_{i-1}^{iter}, to the new base model M_i and then fine-tune it. Next, we compute a new diff vector between the fine-tuned model and the current base model M_i. This diff vector is then carried forward to the next model version for fine-tuning in the subsequent iteration.

We denote our iterative recycling-then-finetuning approach as Δ_{iter} and compare it to Δ_{dir}, a direct recycling-then-finetuning approach that applies the diff vector from the latest model version directly to the current model. For training, we follow the procedure outlined in §3.2.

Algorithm 1 Iterative recycling-then-finetuning

Notation: FT denotes fine-tuning.
Input: Base models $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$
Output: Fine-tuned $\mathcal{M}_1^*, \mathcal{M}_2^*, \ldots, \mathcal{M}_n^*$
$\mathcal{M}_1^* \leftarrow \text{FT}(\mathcal{M}_1)$
for $i = 2$ to n **do**
  $\Delta_{i-1}^{iter} = \mathcal{M}_{i-1}^* - M_{i-1}$
  $\mathcal{M}_i^* \leftarrow \text{FT}(M_i + \Delta_{i-1}^{iter})$
end for
return $\mathcal{M}_1^*, \mathcal{M}_2^*, \ldots, \mathcal{M}_n^*$

	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	24.4	64.5	65.5
+ Δ^{dir}	62.7	**77.6**	77.2
+ Δ^{iter}	**67.0**	77.3	**77.5**
Direct FT	60.4	75.7	75.6

Table 6: Comparison of direct (Δ^{dir}) and iterative (Δ^{iter}) recycling-then-finetuning. $\mathcal{M}_1$ and $\mathcal{M}_2$'s results are omitted as these models remain identical across approaches (see Algorithm 1). Both methods significantly boost GSM8K performance, surpassing standard fine-tuning without recycling (Direct FT). See Appendix F for results without fine-tuning.

6.2 Results and discussion

Iterative recycling-then-finetuning significantly improves performance: Table 6 shows the performance of our two recycling approaches: direct recycling-then-finetuning (Δ_{dir}) and iterative recycling-then-finetuning (Δ_{iter}). Both approaches lead to significant performance improvements across model versions on GSM8K, with larger gains observed in earlier versions. For instance, Δ_{iter} achieves absolute accuracy improvements of +42.6%, +12.8%, and +12% over $\mathcal{M}_3$, $\mathcal{M}_4$, and $\mathcal{M}_5$, respectively. Both approaches also outperform the standard fine-tuning baseline (without recycling) by a substantial margin. Specifically, Δ_{iter} yields average accuracy improvements of +6.6% on $\mathcal{M}_3$ and +1.9% on $\mathcal{M}_5$ compared to standard fine-tuning. Overall, Δ_{iter} either performs similarly to or significantly better than Δ_{dir} across model versions. These results suggest that in scenarios where the base model is continuously updated, adopting an iterative recycling strategy is beneficial.

Iterative recycling-then-finetuning leads to faster convergence: Figure 3 shows that both recycling approaches—iterative recycling-then-

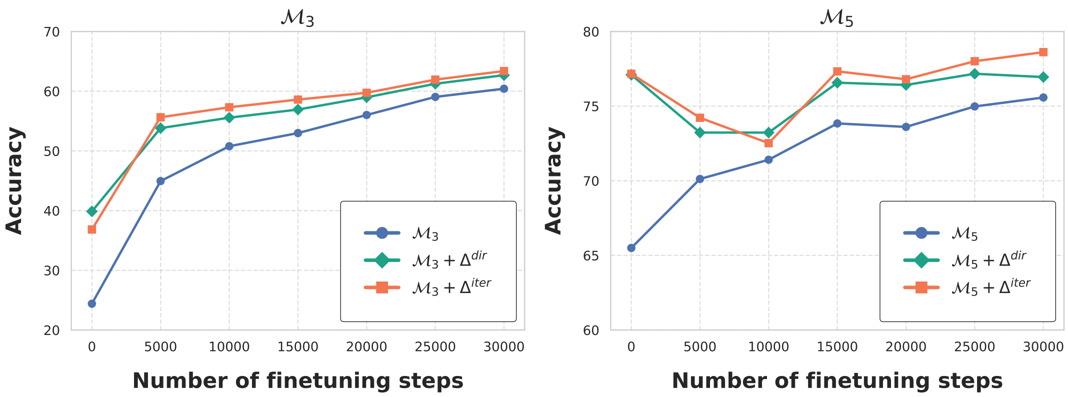

Figure 3: GSM8K performance showing that recycling-then-finetuning (Δ^{dir}) and iterative recycling-then-finetuning (Δ^{iter}) offer faster convergence.

finetuning (Δ_{iter}) and direct recycling-then-finetuning (Δ^{dir})—offer more computationally efficient starting points for fine-tuning. Overall, Δ_{iter} consistently outperforms Δ^{dir} in terms of training efficiency and significantly improves upon standard fine-tuning (without recycling). These findings suggest that iterative recycling not only improves model performance but also boosts training efficiency, effectively leveraging the knowledge encoded in diff vectors across different model versions.

7 Related work

Most prior work focuses on recycling fine-tuning across tasks with the same base model (Vu et al., 2022, 2020; Ilharco et al., 2023; Yadav et al., 2023; Yu et al., 2024), while we explore recycling fine-tuning across different model versions, architectures, and sizes trained on the same data. Previous studies (Lester et al., 2022; Su et al., 2022) also examine recycling fine-tuning across architectures and sizes, but they focus on soft prompts with non-instruction-tuned models, whereas we utilize the diff vectors between model versions. Additionally, some work reuses small models for large ones by duplicating (Chen et al., 2022), progressively stacking (Gong et al., 2019), or merging parameters (Wang et al., 2023). Another line of research suggests that model updates can be transferred in a continual context, where prior knowledge from earlier iterations is used to enhance adaptation and efficiency over time. Qin et al. (2023) explores recyclable tuning in the continual pre-training process, highlighting the benefits of reusing fine-tuned weights when transitioning to an upgraded model. Our work differs by providing a comprehensive evaluation of model update recycling in a model development setup, specifically focusing on reusing fine-tuned updates across different model versions to improve LLM training efficiency.

8 Conclusion

In this paper, we present a novel approach for recycling fine-tuning across different versions of LLMs, addressing the inefficiencies of retraining when new base models are introduced. Our method leverages diff vectors to transfer fine-tuning updates, significantly reducing the need for repeated fine-tuning while preserving competitive performance. Empirical evaluations across various open-weight model versions confirm the effectiveness of this approach, particularly when the source and target models are linearly connected. Additionally, we demonstrate its applicability in multilingual model development and show that recycled fine-tuning serves as a strong initialization for further training, accelerating convergence and improves performance. We further extend our approach to continuous model development, where iterative recycling progressively enhances performance while minimizing computational costs. Our results establish recycling fine-tuning as a practical and efficient strategy for sustaining high-quality LLM updates with reduced training overhead, paving the way for more sustainable AI model development. We hope that our approach can help the community keeps pace with the rapid advancements in LLM development.

9 Limitations

Our experiments focus on evaluating supervised fine-tuning as a post-training method, using math reasoning instruction data for fine-tuning. To improve generalization, it is important to explore a broader range of downstream tasks and post-training techniques, such as reinforcement learning with human feedback (RLHF), across different LLM capabilities. Expanding our approach to encompass more aspects of model development offers a promising direction for further exploration.

10 Ethical considerations and risks

Our approach aims to improve the efficiency of LLM development by reducing the need for extensive fine-tuning. However, this method carries certain risks. One concern is that reusing fine-tuning updates may inadvertently transfer biases or undesirable behaviors from one model to another, especially if the source model contains such issues.

Although this approach lowers computational costs, it does not negate the need for careful model design to ensure ethical behavior. Thus, responsible implementation of this technique is crucial. Future research should explore its ethical implications across different model architectures and training approaches.

References

Duygu Altinok. 2024. Instructurca: A diverse instructional content dataset for turkish.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2022. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. 2024. Olmes: A standard for language model evaluations. *arXiv preprint arXiv:2406.08446*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021b. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021c. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. 2024a. Tülu 3: Pushing frontiers in open language model post-training.

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

Brian Lester, Joshua Yurtsever, Siamak Shakeri, and Noah Constant. 2022. Reducing retraining by recycling parameter-efficient prompts. *arXiv preprint arXiv:2208.05577*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *Preprint*, arXiv:2206.14858.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *The Eleventh International Conference on Learning Representations*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

Yujia Qin, Cheng Qian, Xu Han, Yankai Lin, Huadong Wang, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. Recyclable tuning for continual pre-training. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11403–11426.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David I Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, et al. 2024a. Global mmlu: Understanding and addressing cultural and linguistic biases in multilingual evaluation. *arXiv preprint arXiv:2412.03304*.

Shivalika Singh, Freddie Vargus, Daniel D'souza, Börje Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura O'Mahony, Mike Zhang, Ramith Hettiarachchi, Joseph Wilson, Marina Machado, Luisa Moura, Dominik Krzemiński, Hakimeh Fadaei, Irem Ergun, Ifeoma Okoh, Aisha Alaagib, Oshan Mudannayake, Zaid Alyafeai, Vu Chien, Sebastian Ruder, Surya Guthikonda, Emad Alghamdi, Sebastian Gehrmann, Niklas Muennighoff, Max Bartolo, Julia Kreutzer, Ahmet Üstün, Marzieh Fadaee, and Sara Hooker. 2024b. Aya dataset: An open-access collection for multilingual instruction tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11521–11567.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3949–3969.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. 2020. Exploring and predicting transferability across NLP tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7882–7926.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. 2023. Learning to grow pretrained models for efficient transformer training. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, pages 7093–7115.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 57755–57775. PMLR.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *Preprint*, arXiv:2304.06364.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Additional results for recycling fine-tuning

We provide the evaluation from Tulu 3 and OLMo2, inclduing results on mathematical (GSM8K, MATH), reasoning skills (ARC Challenge and GPQA) and general knowledge (MMLU), instruction-following abilities (IFEval), shown in Table 7. In the following, we specifically discuss the evaluation details.

Models	GSM8K	MATH	ARC_C	GPQA	MMLU	IFEval
Tülu 3 8B SFT	76.2	31.6	79.1	31.0	65.1	72.0
Tülu 3 8B DPO	84.1	42.4	79.6	33.3	68.4	81.7
Tülu 3 8B Instruct	87.9	43.4	79.4	34.4	67.9	81.5
Llama 3.0 8B	55.6	17.3	79.7	22.3	66.7	76.6
+ Δ_{SFT}	71.8	26.3	77.9	32.1	63.5	69.1
+ Δ_{DPO}	81.1	38.1	78.6	31.9	67.5	82.9
+ Δ_{IT}	85.1	37.6	79.1	32.4	66.2	82.4
OLMo 2 7B	67.2	19.2	79.9	20.5	63.6	23.0
OLMo 2 7B SFT	71.7	25.2	79.7	27.9	61.2	67.7
OLMo 2 7B DPO	82.5	31.3	80.5	30.6	62.1	73.2
OLMo 2 7B Instruct	85.3	29.7	80.6	29.7	63.3	75.6
OLMo 2 Initial	2.5	1.6	25.7	18.1	25.0	12.2
+ Δ_{SFT}	2.2	0.8	23.8	1.3	1.4	13.7
+ Δ_{DPO}	2.1	0.8	24.1	1.1	1.3	13.7
+ Δ_{IT}	2.0	0.8	24.1	0.6	1.4	13.3
OLMo 2 Stage 1	24.4	5.7	72.7	15.4	59.8	15.7
+ Δ_{SFT}	31.7	8.4	74.3	24.8	55.4	51.4
+ Δ_{DPO}	40.4	9.3	75.0	29.9	56.6	68.0
+ Δ_{IT}	40.2	10.3	75.1	29.9	56.7	68.3
OLMo 2 Final	63.7	17.5	78.6	22.5	62.6	16.1
+ Δ_{SFT}	71.1	23.7	79.0	28.3	59.7	64.3
+ Δ_{DPO}	79.9	27.8	79.3	29.0	63.1	72.6
+ Δ_{IT}	82.8	27.7	79.3	27.2	62.2	72.1

Table 7: Evaluation results on mathematical (GSM8K, MATH), reasoning (ARC Challenge, GPQA), general knowledge (MMLU), and instruction-following (IFEval) abilities. OLMo 2 Initial, OLMo 2 Stage 1, and OLMo 2 Final represent different versions at various stages of the mid-training phase..

B Evaluation details

Our evaluation follows standard practices from prior works and established evaluation tool. We divide the evaluation process into two categories: (1) LLaMA-based evaluations, which follow configurations used in prior LLaMA model assessments, and (2) Olmo2 and Tülu3-based evaluations, which adhere to the evaluation setup from Tülu3 model development. We ensure consistency in shot configuration (zero-shot, few-shot), chain-of-thought (CoT) prompting, and answer extraction methodologies. Below, we provide details for each benchmark.

GSM8K We use an 8-shot CoT setup as in Wei et al. (2023), with greedy sampling. The final numerical value in the response is extracted as the predicted answer. The maximum generation length is 1024 tokens. The same 8-shot CoT evaluation is applied, following the Tülu 3 (Lambert et al., 2024a) methodology with identical answer extraction procedures.

MATH Pre-trained models follow a 4-shot setup based on Lewkowycz et al. (2022), with a maximum generation length of 512 tokens. Post-trained models are evaluated using a 0-shot CoT prompt, enhanced with symbolic computation (`sympy`) for answer validation. Complex expressions are resolved using an equality template with a judge, and the maximum generation length is 5120 tokens. Evaluation remains consistent with the Tülu 3 setup, using a 4-shot CoT approach and a flexible answer extraction strategy to handle formatting inconsistencies.

ARC-Challenge We follow the official evaluation setups: 25-shot for LLaMA pretrained models, 0-shot for instruction-tuned models, and 5-shot for

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	**14.6**	**11.6**	11.4	11.6	16.6
+ Δ_1		8.8	**17.8**	19.2	15.6
+ Δ_2	7.6		12.6	14.6	14.4
+ Δ_3	8.0	9.4		23.4	27.8
+ Δ_4	7.8	8.0	9.8		**34.2**
+ Δ_5	8.0	7.4	11.2	**30.6**	
Direct FT	45.1	50.7	60.4	75.7	75.5

Table 8: MATH500 accuracies also demonstrate that strong models, e.g. $\mathcal{M}_4$ and $\mathcal{M}_5$ are better at utilizing recycled fine-tuning.

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	14.6	11.6	11.4	11.6	16.6
+ $\Delta_1 \rightarrow$ FT		21.0$_{+12.2}$	23.0$_{+5.2}$	**32.0**$_{+12.8}$	34.2$_{+18.6}$
+ $\Delta_2 \rightarrow$ FT	16.2$_{+8.6}$		**26.2**$_{+13.6}$	31.6$_{+17.0}$	31.0$_{+16.6}$
+ $\Delta_3 \rightarrow$ FT	**18.4**$_{+10.4}$	21.2$_{+11.8}$		31.0$_{+7.6}$	34.0$_{+6.2}$
+ $\Delta_4 \rightarrow$ FT	17.4$_{+9.6}$	19.0$_{+11.0}$	23.8$_{+14.0}$		**36.2**$_{+2.0}$
+ $\Delta_5 \rightarrow$ FT	17.2$_{+9.2}$	**21.4**$_{+14.0}$	25.0$_{+13.8}$	30.4$_{-0.2}$	
Direct FT	13.4	17.6	21.6	31.4	33

Table 9: MATH500 accuracies provide another evidence to support recycling fine-tuning provides a stronger starting point for fine-tuning (FT).

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	23.7	24.2	23.2	26.2	25.2
+ $\Delta_1 \rightarrow$ FT		25.2$_{+1.0}$	25.1$_{+1.9}$	**33.3**$_{+7.1}$	25.7$_{+0.5}$
+ $\Delta_2 \rightarrow$ FT	**27.7**$_{+4.0}$		25.2$_{+2.0}$	30.8$_{+4.6}$	27.2$_{+2.0}$
+ $\Delta_3 \rightarrow$ FT	**27.7**$_{+4.0}$	**27.7**$_{+3.5}$		23.7$_{-2.5}$	23.2$_{-2.0}$
+ $\Delta_4 \rightarrow$ FT	24.7$_{+1.0}$	24.7$_{+0.5}$	26.2$_{+3.0}$		**28.2**$_{+3.0}$
+ $\Delta_5 \rightarrow$ FT	26.7$_{+3.0}$	26.7$_{+2.5}$	23.2$_{+0.0}$	25.7$_{-0.5}$	
Direct FT	25.7	26.7	**26.7**	19.1	26.2

Table 10: GPQA$_{\text{Diamond}}$ results from the direct recycling-then-finetuning approach.

Tülu3 and OLMo2.

GPQA A zero-shot setup is adopted, following the Zhong et al. (2023), where the model selects the correct answer from multiple choices. The same zero-shot evaluation and answer extraction procedure as in Tülu 3 (Lambert et al., 2024a) is used.

MMLU Pre-trained models are evaluated in a 5-shot setting, using the standard MMLU (Hendrycks et al., 2021b) prompt to compute negative log-likelihood (NLL) over answer choices. Post-trained models are tested in both 5-shot and 0-shot settings, with the latter incorporating a CoT prompt where the model generates a reasoning step before answering. The maximum generation length is 10 tokens for 5-shot and 1024 tokens for 0-shot evaluations. Macro average scores are reported unless otherwise specified. Evaluation follows the *Tülu 3* zero-shot CoT approach, ensuring consistency in methodology.

IFEval We use Prompt-level scores and instruction-level strict and loose accuracy are computed, with final results reported as the average across these metrics. The same setup is applied, following the programmatic constraint verification method used in Tülu 3 (Lambert et al., 2024a).

DROP A 3-shot setup is used for pre-trained models, with few-shot examples randomly drawn from the training split. F1 scores are reported, and the maximum generation length is set to 32 tokens. This 3-shot evaluation setup is maintained, with greedy sampling following the Tülu 3 (Lambert et al., 2024a) methodology.

C Results on recycling fine-tuning

Table 9 show additional results for MATH500, further illustrating the impact of fine-tuning transfer across different model versions.

D Results on recycling-then-finetuning

We presented the direct recycling-then-finetuning approach for MATH500 in Table 9, while Table 10 reports our GPQA$_{\text{Diamond}}$ results.

E Training dynamics of recycling-then-finetuning

Figure 4 show the training dynamics of $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_4$.

F Result on iterative recycling-then-finetuning

Table 11 presents the comparison comparison between iterative recycling-then-finetuning (Δ^{iter}) and direct recycling-then-finetuning (Δ^{dir}) on GSM8K.

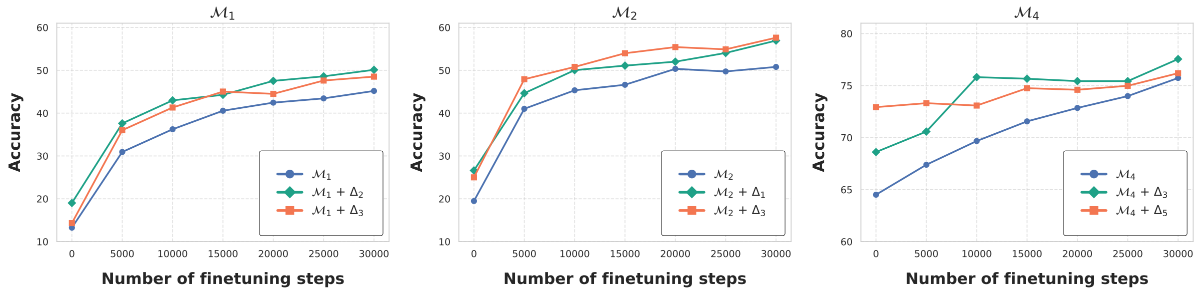

Figure 4: Across different model versions, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_4$, recycling fine-tuning provides a more computationally efficient starting point for fine-tuning on GSM8K.

	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
OLMo 2 7B	24.4	64.5	65.5
+ Δ^{dir}	**39.9**	68.6	77.1
+ Δ^{iter}	36.8	**70.1**	**77.2**

Table 11: Evaluation of merged models using iterative recycling-then-finetuning Δ^{iter} , compared to the direct recycling-then-finetuning approach Δ^{dir} on GSM8K.

FT on Tulu3 Math	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
# tokens	1.2T	2.5T	3.9T	3.9T+26B	3.9T+50B
OLMo 2 7B	13.2	19.4	24.4	64.5	65.5
5K	30.9	41.0	44.9	67.3	70.1
10K	36.2	45.3	50.7	69.6	71.4
15K	40.5	46.6	52.9	71.5	73.8
20K	42.4	50.3	56.0	72.8	73.6
25K	43.4	49.7	59.0	73.9	74.9
30K	45.1	50.7	60.4	75.7	75.5

Table 12: GSM8K performance from fine-tuning with 30K steps. Different OLMo 2 checkpoints are trained on Tülu 3 Persona MATH, GSM, and Algebra265 (220K examples in total).