Efficient Model Development through Fine-tuning Transfer

Anonymous ACL submission

Abstract

Modern LLMs struggle with efficient updates, as each new pretrained model 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. In this paper, we explore the transfer of fine-tuning updates between model versions. Specifically, we derive the *diff vector* (representing the weight changes from fine-tuning) from one *source* model version and apply it to the base model of a different *target* version. Through empirical evaluations on various open-weight model versions, we show that transferring diff vectors can significantly improve the performance of the target base model. For example, transferring the fine-tuning updates from Llama 3.0 8B improves Llama 3.1 8B by 46.9% on IFEval and 15.7% on LiveCodeBench without additional training, even surpassing Llama 3.1 8B Instruct. Furthermore, we demonstrate performance gains on multilingual tasks, with 4.7% and 15.5% improvements on Global MMLU for Malagasy and Turkish, respectively. We observe that these merged models provide stronger initializations for further fine-tuning. Lastly, our controlled experiments suggest that fine-tuning transfer is most effective when source and target models lie in a linearly connected region of parameter space, and we provide a theoretical analysis of our method. Taken together, fine-tuning transfer offers a cost-efficient and practical strategy for continuous LLM development. Our code will be available at https://anonymous.4open.science/r/finetuning-transfer.

1 Introduction

Today's large language models (LLMs) are developed in two stages: (1) *pretraining* on massive corpora with self-supervised learning, and (2) *post-training* with alignment steps (Ouyang et al., 2022; Bai et al., 2022). While this pipeline creates powerful LLMs, it presents a major bottleneck for continuous development: every new version of a pretrained model requires repeating expensive post-training. This challenge is particularly acute in domain- or language-specific applications, where the cost of redoing fine-tuning for each base model update is prohibitive (Qin et al., 2023; Bandarkar et al., 2024).

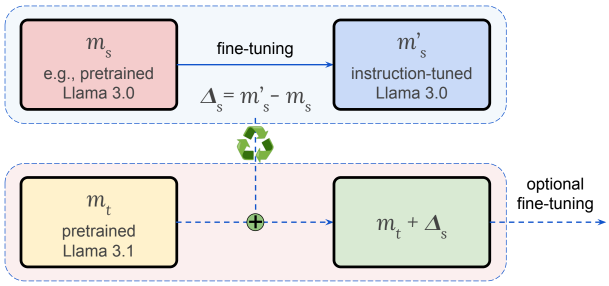

Figure 1: To transfer fine-tuning (e.g., instruction tuning) from a *source* model version s (e.g., Llama 3.0) to a *target* version t (Llama 3.1), we first compute the diff vector $\Delta_s = m'_s - m_s$ from version s, where m'_s is the fine-tuned model (instruction-tuned Llama 3.0) and m_s is the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

In this paper, we explore a method to reduce post-training costs by transferring fine-tuning updates between different model versions. Specifically, we propose incorporating the *weight updates* from a *source* model version s to improve a *target* model version t. Our approach (see Figure 1) first computes the *diff vector* $\Delta_s = m'_s - m_s$ from version s, which represents the difference between the fine-tuned model m'_s (e.g., instruction-tuned) and its base model m_s (pretrained). Intuitively, Δ_s encodes the task-specific updates to the model parameters during fine-tuning, and can be used to transfer knowledge from the source version s to the target version t. Contrary to prior work (Ilharco et al., 2023; Huang et al., 2023), which focuses on improving the capabilities of a single model on a specific target task, we focus on a general-purpose method to transfer updates between different model versions for a variety of downstream tasks. We hypothesize that models fine-tuned using the same or similar training data and procedures exhibit linear relationships across versions: $m'_s - m_s \approx m'_t - m_t$. This suggests that we can approximate the fine-tuned version m'_t of the target base model m_t without training: $m'_t \approx m_t + \Delta_s$. The intuition is supported by linear mode connectivity theory (Mirzadeh et al., 2020; Frankle et al., 2020), which shows that two independently trained networks can be connected by a low-loss path (see Appendix A).

We begin by evaluating the feasibility of our approach through the transfer of *diff vectors* across different versions of open-weight models (Section 2). Recycling the fine-tuning updates from Llama 3.0 yields a 46.9% absolute accuracy improvement on IFEval over Llama 3.1 8B, while also surpassing the performance of Llama 3.1 8B Instruct without additional training.

Motivated by these results, we conduct a case study on the development of multilingual models (Section 3). We observe that diff vectors transfer facilitates a better understanding of the target language. Specifically, transferring weights from a fine-tuned version of Llama 3.0 Instruct to Llama 3.1 Instruct yields absolute accuracy improvements of 4.7% for Malagasy and 15.5% for Turkish on the Global MMLU benchmark (Singh et al., 2024a), without additional training.

To shed light on when fine-tuning transfer is most effective, we perform controlled experiments using OLMo 2's (OLMo et al., 2024) intermediate pretrained checkpoints as different model versions (Section 4). Our results suggest that fine-tuning transfer is most effective when the source and target models lie within a linearly connected region of the parameter space, consistent with linear mode connectivity (Mirzadeh et al., 2020; Ainsworth et al., 2023; Wortsman et al., 2022a,b; Frankle et al., 2020).

Furthermore, we investigate whether the merged model $m_t + \Delta_s$ can serve as a computationally efficient and effective starting point for fine-tuning (Section 5). Our experiments demonstrate that initializing fine-tuning with this merged model can accelerate convergence and improve accuracy compared to training on top of m_t. We find that even when the selected diff vector is suboptimal, fine-tuning the merged model consistently improves performance compared to direct fine-tuning, without harming generalization to unseen tasks. This suggests that fine-tuning transfer can serve as a robust and effective intermediate step when training is feasible.

Lastly, we explore a continuous model development scenario (in Section 6) in which new model versions are regularly released. We propose an iterative recycling–then–fine-tuning approach that incrementally accumulates fine-tuning updates from previous versions. In summary, our key contributions are as follows.

- Introducing an approach for transferring fine-tuning updates between model versions via diff vector transfer.
- Demonstrating that this approach can reduce training costs while maintaining competitive performance.
- Validating the approach in a multilingual model development setting, showing improved language-specific performance without retraining.
- Establishing conditions for effective fine-tuning transfer, particularly when models exhibit linear mode connectivity.
- Proposing a recycling-then-finetuning strategy to improve both efficiency and performance in a continuous model development setting.

2 Transferring fine-tuning updates across model versions

In this section, we explore transferring the weight changes from a source model version s to a target model version t, denoted $\mathcal{T}_{s \to t}$, without additional training. Specifically, we directly merge (add) the diff vector $\Delta_s = m'_s - m_s$ from version s, which captures the parameter adaptations from the base model m_s to its fine-tuned counterpart m'_s, onto the new base model m_t in version t, without any gradient-based training. Our results (Table 1) show that fine-tuning updates can be effectively transferred across model versions, as $m_t + \Delta_s$ often performs comparably to its fine-tuned counterpart m'_t.

2.1 Experimental setup

We conduct experiments with various open-weight models, including Llama (Dubey et al., 2024),

Model	GSM8K	MATH	ARC_C	GPQA	MMLU	IFEval	HE+	MBPP+	LCB	BCB	Avg.
Llama 3.0 8B Instruct	81.1	28.8	82.4	**31.5**	64.9	**76.6**	56.7	**55.6**	14.0	6.8	49.8
Llama 3.0 8B	55.6	17.3	79.7	22.3	66.7	34.5	31.1	51.3	0.0	6.1	34.5
+ $\Delta_{3.1}$	**82.8**	**44.7**	**83.0**	25.9	**70.0**	**76.6**	**62.8**	55.3	**15.8**	**12.8**	**53.0**
Llama 3.1 8B Instruct	**86.5**	**50.3**	**83.8**	31.3	**72.9**	80.5	**61.0**	54.8	16.0	**14.9**	**55.2**
Llama 3.1 8B	56.6	19.3	79.2	21.9	66.8	36.4	29.9	51.9	0.4	5.4	36.8
+ $\Delta_{3.0}$	79.8	29.9	82.9	**32.6**	65.1	**83.3**	55.5	**56.6**	**16.1**	10.1	51.2

Table 1: Fine-tuning transfer significantly improves the performance of the target base model across various tasks, achieving results comparable to its fine-tuned counterpart in many cases. Here, $\Delta_{3.0}$ and $\Delta_{3.1}$ represent the diff vectors between Llama Instruct and Llama for versions 3.0 and 3.1, respectively. Notably, adding the diff vector Δ_s from a different model version can effectively transform a non-instruction-tuned model (e.g., Llama 3.0 or Llama 3.1) into one that follows instructions well (Llama 3.0 + $\Delta_{3.1}$ or Llama 3.1 + $\Delta_{3.0}$) without further training. Additional results for OLMo and Tülu can be found in Appendix B.1, where we additionally find that advanced LLM capabilities, attained through alignment tuning stages such as Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), or Group Relative Policy Optimization (GRPO), can be successfully transferred across different model versions.

OLMo (OLMo et al., 2024), and Tülu (Lambert et al., 2024). Throughout this work, we ensure that our source and target models are of the same architecture. We provide additional cross-architecture transfer results in Appendix B.2 and leave further research on cross-architecture recycling as future work. Our study explores both transfer directions: from an older model version to a newer one (*recycling*) and from a newer version to an older one (*backporting*).

Recycling can save training time and computational resources, while incorporating post-training capabilities into the newer pretrained model. Conversely, *backporting* is beneficial when the older base model is better optimized for a specific use case (e.g., a particular language), allowing the user to take advantage of the new fine-tuning improvements while maintaining optimization and compatibility.[1] We emphasize that our goal is not to achieve state-of-the-art results, but instead to assess the feasibility of transferring fine-tuning updates between model versions.

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., 2021b), reasoning with ARC_C (Clark et al., 2018) and GPQA (Rein et al., 2024), instruction-following with IFEval (Zhou et al., 2023), code generation with HumanEval+ (HE+ in Table 1) and MBPP+ (Liu et al., 2023), LiveCodeBench (Jain et al., 2024), and BigCodeBench (Zhuo et al., 2024) (LCB and BCB in Table 1 respectively). We compare its performance to that of directly fine-tuned m_t (i.e., m'_t).[2] See Appendix C for evaluation details.

2.2 Results and discussion

Transferring fine-tuning substantially boosts the target base model's performance: Table 1 shows our results when transferring fine-tuning (i.e., instruction tuning) updates between Llama 3.0 and Llama 3.1. First, we note that Llama 3.0 Instruct consistently performs better than Llama 3.1 (and vice versa). This highlights that most capabilities of the instruction-tuned model arise post-training. Here, we attempt to transfer such capabilities between model versions, and thus bypass the alignment stage. Strikingly, adding the diff vector Δ_s from a different model version can effectively transform a non-instruction-tuned model (e.g., Llama 3.0 or Llama 3.1) into one that follows instructions well (Llama 3.0 + $\Delta_{3.1}$ or Llama 3.1 + $\Delta_{3.0}$). For example, our approach yields 42.1% and 46.9% absolute accuracy improvements on the instruction-following benchmark IFEval over the base versions of Llama 3.0 and Llama 3.1, respectively. Large gains are also observed across the board on math, code, and reasoning benchmarks, with an average improvement of 18.5% for Llama 3.0 and 14.4% for Llama 3.1. These results suggest that advanced knowledge and instruction-following abilities can be efficiently transferred

[1] In software development, *backporting* refers to the process of adapting features or updates from a newer version of a software system or component for use in an older version.

[2] For evaluating HumanEval+ and MBPP+ we use `EvalPlus` (Liu et al., 2023), and the official evaluation libraries for LiveCodeBench and BigCodeBench. All other tasks are evaluated using the `lm-evaluation-harness` library (Gao et al., 2024).

between model versions without further training. In general, Llama 3.0 benefits more from the backported diff vector $\Delta_{3.1}$ from version 3.1 than Llama 3.1 does from recycling version 3.0's diff vector $\Delta_{3.0}$.

Transferring fine-tuning can achieve performance comparable to the fine-tuned model: Our results demonstrate that the merged model $m_t + \Delta_s$ can perform on par with its fine-tuned counterpart m'_t across various tasks. This is particularly true for Llama 3.0 + $\Delta_{3.1}$, which matches or surpasses Llama 3.0 Instruct on eight out of ten tasks we evaluated. Interestingly, Llama 3.1 + $\Delta_{3.0}$ outperforms LLama 3.1 Instruct on four out of the ten benchmarks. This is a testament to the diff vector's ability to encode advanced reasoning and instruction-following capabilities. Overall, our results suggest that fine-tuning transfer provides an effective and extremely low-cost method to improve model performance when training is prohibitively expensive.

3 Efficient multilingual model development

Motivated by our results in Section 2, we now turn toward applying our fine-tuning transfer approach in a multilingual model development setting. We focus exclusively on a *recycling* scenario, where our aim is to transfer the language-specific instruction tuning updates from an older model version to a newer one.

For language-specific instruction tuning, we fine-tune an instruction-tuned model rather than a pretrained one. This approach aligns with the common practice of using an instruction-tuned English or multilingual model as the foundation when developing language-specific models. A key challenge in this setting is that state-of-the-art LLMs often include multilingual data in pretraining and instruction tuning, which makes it unclear whether language-specific fine-tuning is still necessary. *How effective is our recycling approach when applied to a multilingual instruction-tuned model?* Our results show that recycling fine-tuning remains effective in this scenario, as long as the target base model is outperformed by the fine-tuned model of the source version.

3.1 Experimental setup

We fine-tune Llama 3.0 Instruct (m_s) separately on language-specific instruction tuning data for three languages: Malagasy, Sinhala, and Turkish. We use the Aya dataset (Singh et al., 2024b) for Malagasy (14.6K examples) and Sinhala (14.5K examples), and the InstrucTurca dataset (Altinok, 2024) for Turkish (16.7K examples).[3] Each model is trained for 30K training steps with a learning rate of 5e-6 and a batch size of 8, using 4 NVIDIA A100-80G GPUs.[4]

After training on each language, we compute the diff vector $\Delta_s = m'_s - m_s$ and add it to Llama 3.1 Instruct m_t. We simulate a low-resource setting and do not perform any additional training with language-specific data. The merged model $m_t + \Delta_s$ is evaluated against the base model m_t on the Global MMLU benchmark (Singh et al., 2024a).

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 2: Recycling fine-tuning updates improves multilingual performance on Global MMLU without retraining, yielding a 4.7% and 15.5% absolute improvement for Malagasy and Turkish, respectively, compared to Llama 3.1 8B Instruct. $\Delta_{3.0}$ represents the diff vector between Llama 3.0 Instruct and its monolingual fine-tuned (FT) version.

3.2 Results and discussion

Transferring fine-tuning is effective for developing multilingual models: Our results in Table 2 demonstrate the benefits of reusing fine-tuning updates in multilingual model development. For Malagasy and Turkish, transferring the diff vector from Llama version 3.0 to 3.1 results in significant accuracy improvements (4.7% and 15.5%, respectively) over Llama 3.1 8B Instruct. Our recycling approach performs better than the fine-tuned Llama 3.0 Instruct model for Malagasy (1.5% accuracy improvement) and maintains similar performance for Turkish.

On the other hand, for Sinhala, recycling fine-tuning offers no advantage, as Llama 3.1 Instruct already outperforms the previously fine-tuned Llama

[3]To simulate a low-resource setting, we sampled 6.5% of the original InstrucTurca dataset, which contains 2.58 million examples, resulting in approximately 16.7K examples.

[4]We use the AdamW optimizer with a linear scheduler and a warmup ratio of 0.03. We disable dropout and exclude weight decay for embeddings. The sequence length is 2048. We use `open-instruct` (Lambert et al., 2024) for training and `lm-evaluation-harness` (Gao et al., 2024) for evaluation.

3.0 Instruct. However, even in this case, recycling does not significantly reduce performance.

4 When is fine-tuning transfer effective?

Having demonstrated the effectiveness of fine-tuning transfer, 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 assess the impact of transferring fine-tuning updates between them. Our results reveal that fine-tuning transfer is most successful when the source and target models are close within a linearly connected region of the parameter space, consistent with linear mode connectivity. We provide further theoretical analysis in Appendix A.

4.1 Experimental setup

We conduct experiments with the publicly available intermediate checkpoints of OLMo 2 7B.[5] The base OLMo 2 model was trained in two stages: (1) a general web-based pretraining stage (stage 1), and (2) a mid-training stage (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, at 600K steps), $\mathcal{M}_3$ (end-stage 1, at 929K steps), $\mathcal{M}_4$ (mid-stage 2, at 6K steps), and $\mathcal{M}_5$ (end-stage 2, at 12K steps). Each $\mathcal{M}_i$ is treated as a distinct model version. We investigate both transfer scenarios: (1) recycling ($\mathcal{T}_{\mathcal{M}_i \to \mathcal{M}_j}, i < j$), and (2) backporting ($\mathcal{T}_{\mathcal{M}_j \to \mathcal{M}_i}, j > i$).

Due to our limited computational resources, supervised fine-tuning with a large instruction tuning dataset would be prohibitively expensive. We therefore fine-tune all model versions using a subset of the math reasoning instruction tuning data from Tülu 3, which includes Tülu 3 Persona MATH, GSM, and Algebra (220K examples total), following the training procedure described in Section 3.1.

We evaluate our models on GSM8K and the MATH500 subset (Hendrycks et al., 2021b) of the MATH dataset. These datasets are selected because fine-tuning on Tülu 3's math reasoning data significantly improves performance on them, allowing for a clearer analysis of the impact of transferring fine-tuning updates between model versions.[6]

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

[6] For evaluation, we use the OLMES library (Gu et al., 2024).

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	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**	
FT($\mathcal{M}_i$)	45.1	50.7	60.4	75.7	75.5

Table 3: GSM8K accuracies indicating that more powerful models are better at leveraging transferred fine-tuning. Effective use of transferred fine-tuning only emerges once the target base model reaches a certain level of capability. Furthermore, fine-tuning transfer works best when the source and target models are close within a linearly connected region of the parameter space. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_i refers to the diff vector resulting from the fine-tuning of version i. FT($\mathcal{M}_i$) denotes applying fine-tuning directly to $\mathcal{M}_i$. See Table 13 in Appendix D for MATH500 results.

4.2 Results and discussion

More powerful models are better at leveraging transferred fine-tuning: Our results in Table 3 indicate that stronger models are more effective at leveraging transferred fine-tuning updates. While transferring fine-tuning can improve performance for $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, the merged models $\mathcal{M}_i + \Delta_j$ (Δ_j denotes the diff vector from model version $\mathcal{M}_j$, $j \neq i$) still fall significantly short of their fine-tuned counterparts, denoted FT($\mathcal{M}_i$). On GSM8K, the accuracy gaps between the best $\mathcal{M}_i + \Delta_j$ and FT($\mathcal{M}_i$) are 26.1%, 24.1%, 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 $\mathcal{M}_5$ (i.e., $\mathcal{M}_5 + \Delta_4$) surpasses fine-tuning directly on $\mathcal{M}_5$ (FT($\mathcal{M}_5$)), achieving 1.6% accuracy improvement (77.1% vs. 75.5%). Similar trends are observed on MATH500. This pattern suggests an emergent ability—effective use of transferred fine-tuning only emerges when the target base model is sufficiently strong. In other words, the benefits of transferring fine-tuning only become significant beyond a certain level of capability.

Fine-tuning transfer works best when models are close in the parameter space: Our results also suggest that fine-tuning transfer is most effective when the source and target models are closely connected in the parameter space. On both GSM8K and MATH500, models $\mathcal{M}_1$ and $\mathcal{M}_2$ benefit more from Δ_3 than from Δ_4 or Δ_5. 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, transferring between 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.[7]

5 Fine-tuning transfer as a starting point for further fine-tuning

So far, we have explored a scenario where fine-tuning updates are transferred between model versions without additional fine-tuning. We now switch gears to investigate whether the merged model $m_t + \Delta_s$ can serve as a stronger and more computationally efficient starting checkpoint for further fine-tuning. We conduct controlled experiments comparing two approaches: fine-tuning the merged model $m_t + \Delta_s$ versus fine-tuning m_t directly. 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 suggests that fine-tuning transfer can be a useful intermediate step when additional training is feasible. We refer to this approach as *"transferring-then-finetuning"*.

5.1 Experiment setup

We follow the training procedure outlined in Section 3.1. For evaluation, we use GSM8K and MATH500, along with an additional dataset to assess how well our transferring-then-finetuning approach generalizes to the unseen task $\text{GPQA}_{\text{Diamond}}$ (Rein et al., 2024).

5.2 Results and discussion

Transferring-then-finetuning can substantially boost performance: Our results are summarized in Table 4. Transferring-then-finetuning offers significant improvements over our vanilla transfer approach (without additional fine-tuning) 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. The benefits are most pronounced for weaker base models ($\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$) across all diff vectors, as well as for stronger base models when paired with a weak diff vector (e.g., $\mathcal{M}_5 + \Delta_1$).

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	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	$48.5_{+34.2}$	$\mathbf{57.6}_{+32.6}$		$77.6_{+9.0}$	$\mathbf{77.1}_{+6.8}$
$+ \Delta_4 \rightarrow$ FT	$48.2_{+36.4}$	$56.7_{+38.7}$	$\mathbf{63.7}_{+41.1}$		$77.0_{-0.1}$
$+ \Delta_5 \rightarrow$ FT	$47.6_{+35.7}$	$55.6_{+39.6}$	$63.5_{+39.5}$	$74.6_{+1.7}$	
FT($\mathcal{M}_i$)	45.1	50.7	60.4	75.7	75.5

Table 4: GSM8K accuracies showing that fine-tuning transfer provides a stronger starting point (i.e., $\mathcal{M}_i + \Delta_j$) for further fine-tuning (FT). Numbers in subscript indicate improvement over the baseline without fine-tuning. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_i refers to the diff vector resulting from the fine-tuning of version i. FT($\mathcal{M}_i$) denotes applying fine-tuning directly to $\mathcal{M}_i$. See Table 14 in Appendix E for MATH500 results.

Interestingly, fine-tuning also helps bridge the performance gap between the merged models $\mathcal{M}_i + \Delta_j$ ($j \neq i$) for each base model $\mathcal{M}_i$. For example, fine-tuning dramatically improves the performance of $\mathcal{M}_5 + \Delta_1$ by 59% and $\mathcal{M}_5 + \Delta_2$ by 61.4%, closing the gap with the fine-tuned versions of $\mathcal{M}_5 + \Delta_3$ and $\mathcal{M}_5 + \Delta_4$. This reduces the need to preselect the best diff vector when multiple choices are available. Importantly, transferring-then-finetuning generally outperforms standard fine-tuning regardless of the diff vector used.

Transferring-then-finetuning can offer faster convergence: Figure 2 shows that using the merged model $\mathcal{M}_i + \Delta_j$ as the initial checkpoint improves training efficiency. Specifically, $\mathcal{M}_i + \Delta_j$ not only converges significantly faster than $\mathcal{M}_i$ during fine-tuning but also reaches a higher peak accuracy on GSM8K. Overall, our results suggest that transferring-then-finetuning is a cost-effective approach that reduces the number of fine-tuning steps, thereby improving training efficiency.

Transferring-then-finetuning does not negatively impact model generalization: As shown in Table 5, this approach attains strong zero-shot generalization on the unseen task $\text{GPQA}_{\text{Diamond}}$, comparable to standard fine-tuning. These results suggest that transferring-then-finetuning does not lead to overfitting, demonstrating its broad applicability across diverse tasks.

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

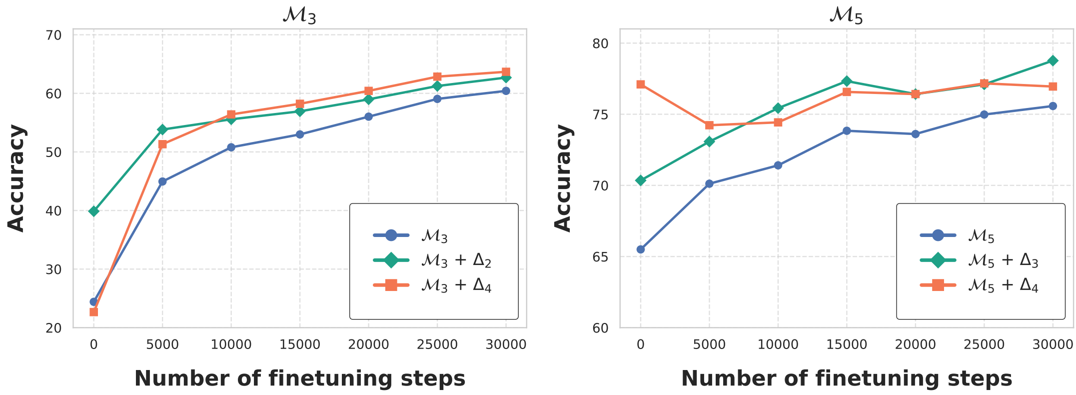

Figure 2: GSM8K performance showing that fine-tuning transfer provides a more computationally efficient starting point (i.e., $\mathcal{M}_i + \Delta_j$) for further training. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_j refers to the diff vector resulting from the fine-tuning of version j. Additional results for $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_4$ can be found Appendix E.

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	23.7	24.2	23.2	26.3	25.3
+ $\Delta_1 \rightarrow$ FT		$25.3_{-1.0}$	$25.2_{-2.1}$	$\mathbf{33.3}_{+9.6}$	$25.8_{-0.5}$
+ $\Delta_2 \rightarrow$ FT	$\mathbf{27.8}_{-1.5}$		$25.3_{+0.0}$	$30.8_{+6.6}$	$\mathbf{27.3}_{+3.1}$
+ $\Delta_3 \rightarrow$ FT	$\mathbf{27.8}_{-0.5}$	$\mathbf{27.8}_{+0.5}$		$23.7_{+0.5}$	$\mathbf{27.3}_{+5.1}$
+ $\Delta_4 \rightarrow$ FT	$24.8_{-2.0}$	$24.8_{-4.5}$	$26.3_{+2.1}$		$24.2_{-1.1}$
+ $\Delta_5 \rightarrow$ FT	$22.7_{-5.1}$	$26.8_{+0.0}$	$23.2_{-1.0}$	$27.8_{+4.6}$	
FT($\mathcal{M}_i$)	25.8	26.8	**26.8**	19.2	26.3

Table 5: $\text{GPQA}_{\text{Diamond}}$ accuracies showing that fine-tuning transfer provides a stronger starting point (i.e., $\mathcal{M}_i + \Delta_j$) for further fine-tuning (FT), and transferring-then-finetuning does not negatively impact model generalization to unseen tasks. Numbers in subscript indicate improvement over the baseline without fine-tuning. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_j refers to the diff vector resulting from the fine-tuning of version j. FT($\mathcal{M}_i$) denotes applying fine-tuning directly to $\mathcal{M}_i$.

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

Building on the insights from our previous experiments, we now explore a continuous model development setting in which new versions of a pretrained model are regularly released. At the core of 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 previous diff vectors iteratively. Specifically, 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—$\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$, $\mathcal{M}_5$ (described in Section 4.1) 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 $\mathcal{M}_i$, and then further fine-tune the resulting model. Next, we compute a new diff vector between the fine-tuned model and the current base model $\mathcal{M}_i$. This new diff vector is then carried forward to the next model version for fine-tuning in the subsequent iteration.

We refer to our iterative recycling-then-finetuning approach as Δ^{iter} and compare it to Δ^{dir}, the direct recycling-then-finetuning approach as described in 5. We follow the training procedure outlined in Section 3.1.

6.2 Results and discussion

Iterative recycling-then-finetuning significantly improves performance: Table 6 compares the performance of two recycling approaches: iterative recycling-then-finetuning (Δ^{iter}) and direct recycling-then-finetuning (Δ^{dir}). Both approaches lead to significant performance improvements across model versions on GSM8K, with larger gains observed in earlier versions. Both approaches outperform the standard fine-tuning baseline (without recycling) by a substantial margin.

	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	24.4	64.5	65.5
+ Δ^{dir} $\rightarrow$ FT	$62.7_{+38.3}$	$\mathbf{77.6}_{+13.1}$	$77.0_{+11.5}$
+ Δ^{iter} $\rightarrow$ FT	$\mathbf{63.4}_{+39.0}$	$77.4_{+12.9}$	$\mathbf{78.6}_{+13.1}$
FT($\mathcal{M}_i$)	60.4	75.7	75.6

Table 6: Both iterative (Δ^{iter}) and direct (Δ^{dir}) recycling-then-finetuning approaches significantly boost GSM8K performance, surpassing standard fine-tuning without recycling (FT($\mathcal{M}_i$)). Numbers in subscripts indicate improvement over OLMo 2 7B checkpoints. At a high level, Δ^{iter} gradually incorporates fine-tuning updates, i.e., diff vectors, from previous model versions, while Δ^{dir} directly applies the diff vector from the latest model version to the current model. Results for $\mathcal{M}_1$ and $\mathcal{M}_2$ are omitted as these models remain identical across the two approaches. See Appendix F for additional results.

In general, Δ^{iter} performs similarly to or better than Δ^{dir} across all model versions. These results suggest that in scenarios where the base model is continuously updated, adopting an iterative recycling strategy is beneficial and does not result in error propagation.

7 Related work

Fine-tuning transfer: Prior work has studied how to reuse fine-tuning updates on a fixed base model to improve generalization across tasks, domains, and languages. This includes full-model adaptation (Phang et al., 2018; Pruksachatkun et al., 2020; Vu et al., 2020, 2021; Aghajanyan et al., 2021) as well as parameter-efficient modules such as adapters (Pfeiffer et al., 2021; Poth et al., 2021), soft prompts (Vu et al., 2022b,a; Su et al., 2022; Asai et al., 2022), and LoRA matrices (Huang et al., 2024; Zadouri et al., 2024; Ostapenko et al., 2024); see Yadav et al. (2024a) for a comprehensive survey. These methods typically assume a shared base model and focus on transferring capabilities across tasks or domains. Similarly, model merging combines multiple task-specific models based on the same model to create a more powerful model (Ilharco et al., 2023; Yadav et al., 2023; Wang et al., 2024a; Ramé et al., 2024; Yu et al., 2024; Yadav et al., 2024b; Ahmadian et al., 2024; Bandarkar et al., 2025). Recent work also extrapolates RLHF updates back to the base model (Zheng et al., 2024; Lin et al., 2025). In contrast, our work focuses on transferring fine-tuning updates *across different model versions*, addressing the challenge of frequent model upgrades in LLM development.

Cross-model fine-tuning transfer: Several studies investigate transferring fine-tuning across different model architectures, primarily focusing on lightweight modules in non-instruction-tuned settings (Lester et al., 2022; Su et al., 2022; Wang et al., 2024b; Fleshman and Van Durme, 2024; Echterhoff et al., 2024).

Closely related to our work, Qin et al. (2023) study recyclable fine-tuning in a continual domain adaptation setting from the BERT (Devlin et al., 2019) era, where fine-tuning updates from domain-adapted checkpoints are reused to adapt to new domains. Other efforts aim to reuse weights across divergent model architectures through duplication (Chen et al., 2022), progressive stacking (Gong et al., 2019), or parameter merging (Wang et al., 2023). While these works reuse fine-tuning updates across domains, skills, or architectures, our work focuses on transferring full fine-tuning updates across different versions of both pretrained and instruction-tuned LLMs. This enables efficient model development even when the underlying models differ in pretraining scale or alignment steps. We evaluate both recycling and backporting scenarios. Our approach complements prior work, and combining these directions presents a promising avenue for future research.

8 Conclusion

Our study demonstrates that fine-tuning transfer offers a practical approach to mitigate the inefficiencies of frequent model updates. By applying diff vectors from a fine-tuned source model version to a different target model version, we achieve substantial performance improvements without the need for full fine-tuning. In a multilingual context, this approach can significantly boost performance on target-language tasks, offering an efficient solution for language-specific fine-tuning. Through controlled experiments, we show that fine-tuning transfer is most effective when the source and target models are linearly connected in the parameter space. Furthermore, this approach can offer a more robust and computationally efficient starting checkpoint for further fine-tuning. Taken together, we hope that our work will spur more fundamental research into the efficient development of modern LLMs.

Limitations

Our experiments focus on evaluating supervised fine-tuning as a post-training method, using math reasoning instruction data. However, supervised fine-tuning is only one part of the broader post-training process. Modern LLMs often undergo multiple post-training stages, including reinforcement learning with human feedback (RLHF), preference optimization, or training-then-merging techniques. It is also important to evaluate a broader range of downstream tasks to better assess generalization across different LLM capabilities. In addition, the impact of model shift, such as weight movement, changes in the loss landscape, or representational drift, on the transferability of diff vectors remains underexplored. Expanding our approach to cover these aspects of model development is a promising direction for future work.

Ethical considerations and risks

Our approach aims to improve the efficiency of LLM development by reducing the need for extensive alignment process. 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

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5799–5811.

Arash Ahmadian, Seraphina Goldfarb-Tarrant, Beyza Ermis, Marzieh Fadaee, Sara Hooker, and 1 others. 2024. Mix data or merge models? optimizing for diverse multi-task learning. *arXiv preprint arXiv:2410.10801*.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2023. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*.

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

Akari Asai, Mohammadreza Salehi, Matthew Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6655–6672.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, and 1 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Lucas Bandarkar, Benjamin Muller, Pritish Yuvraj, Rui Hou, Nayan Singhal, Hongjiang Lv, and Bing Liu. 2024. Layer swapping for zero-shot cross-lingual transfer in large language models. *arXiv preprint arXiv:2410.01335*.

Lucas Bandarkar, Benjamin Muller, Pritish Yuvraj, Rui Hou, Nayan Singhal, Hongjiang Lv, and Bing Liu. 2025. Layer swapping for zero-shot cross-lingual transfer in large language models. In *The Thirteenth International Conference on Learning Representations*.

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.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias

Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jessica Echterhoff, Fartash Faghri, Raviteja Vemulapalli, Ting-Yao Hu, Chun-Liang Li, Oncel Tuzel, and Hadi Pouransari. 2024. Muscle: A model update strategy for compatible llm evolution. *arXiv preprint arXiv:2407.09435*.

William Fleshman and Benjamin Van Durme. 2024. Readapt: Reverse engineered adaptation of large language models. *arXiv preprint arXiv:2405.15007*.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

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

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. Lorahub: Efficient cross-task generalization via dynamic loRA composition. In *First Conference on Language Modeling*.

Shih-Cheng Huang, Pin-Zu Li, Yu-Chi Hsu, Kuang-Ming Chen, Yu Tung Lin, Shih-Kai Hsiao, Richard Tzong-Han Tsai, and Hung-yi Lee. 2023. Chat vector: A simple approach to equip llms with instruction following and model alignment in new languages. *arXiv preprint arXiv:2310.04799*.

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*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. 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*.

Yiguan Lin, Bin Xu, Yinghao Li, and Yang Gao. 2025. Extrapolation merging: Keep improving with extrapolation and merging. *arXiv preprint arXiv:2503.04834*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. 2020. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, and 1 others. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. 2024. Towards modular llms by building and reusing a library of loras. *arXiv preprint arXiv:2405.11157*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Clifton Poth, Jonas Pfeiffer, Andreas Rücklé, and Iryna Gurevych. 2021. What to pre-train on? Efficient intermediate task selection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10585–10605.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R. Bowman. 2020. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5231–5247.

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.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741.

Alexandre Ramé, Johan Ferret, Nino Vieillard, Robert Dadashi, Léonard Hussenot, Pierre-Louis Cedoz, Pier Giuseppe Sessa, Sertan Girgin, Arthur Douillard, and Olivier Bachem. 2024. Warp: On the benefits of weight averaged rewarded policies. *arXiv preprint arXiv:2406.16768*.

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*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David I Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, and 1 others. 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, and 14 others. 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, Aditya Barua, Brian Lester, Daniel Cer, Mohit Iyyer, and Noah Constant. 2022a. Overcoming catastrophic forgetting in zero-shot cross-lingual generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9279–9300.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022b. 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, Minh-Thang Luong, Quoc Le, Grady Simon, and Mohit Iyyer. 2021. STraTA: Self-training with task augmentation for better few-shot learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5715–5731.

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.

Ke Wang, Nikolaos Dimitriadis, Alessandro Favero, Guillermo Ortiz-Jimenez, Francois Fleuret, and Pascal Frossard. 2024a. Lines: Post-training layer scaling prevents forgetting and enhances model merging. *arXiv preprint arXiv:2410.17146*.

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*.

Runqian Wang, Soumya Ghosh, David Cox, Diego Antognini, Aude Oliva, Rogerio Feris, and Leonid Karlinsky. 2024b. Trans-lor: towards data-free transferable parameter efficient finetuning. *arXiv preprint arXiv:2405.17258*.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022a. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. 2022b. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971.

Prateek Yadav, Colin Raffel, Mohammed Muqeeth, Lucas Caccia, Haokun Liu, Tianlong Chen, Mohit Bansal, Leshem Choshen, and Alessandro Sordoni. 2024a. A survey on model moerging: Recycling and routing among specialized experts for collaborative learning. *arXiv preprint arXiv:2408.07057*.

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.

Prateek Yadav, Tu Vu, Jonathan Lai, Alexandra Chronopoulou, Manaal Faruqui, Mohit Bansal, and Tsendsuren Munkhdalai. 2024b. What matters for model merging at scale? *arXiv preprint arXiv:2410.03617*.

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.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermis, Acyr Locatelli, and Sara Hooker. 2024. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Chujie Zheng, Ziqi Wang, Heng Ji, Minlie Huang, and Nanyun Peng. 2024. Model extrapolation expedites alignment. *arXiv preprint arXiv:2404.16792*.

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*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

Appendix

A Theoretical justification for Section 2: Transferring fine-tuning updates across model versions

We provide the theoretical motivation for fine-tuning transfer. Let m_s and m_t denote the source and target base models, respectively. Here we assume that m_s and m_t share the same architecture. Let m'_s and m'_t be the fine-tuned versions of m_s and m_t on dataset $\mathcal{D}$. We define $\Delta_s = m'_s - m_s$ as the fine-tuning updates, and hypothesize that Δ_s represents task-specific knowledge that is transferable across model versions.

Linear Mode Connectivity Interpretation. Following linear mode connectivity (Frankle et al., 2020; Mirzadeh et al., 2020; Neyshabur et al., 2020), we assume that m'_s and m'_t (which share the same architecture) arrive at local minima that are connected by a linear path of non-increasing error. Consider some model on this path $m(\lambda)$ given by

$$m(\lambda) = (1-\lambda)m'_s + \lambda m'_t. \quad (1)$$

Substituting m'_s by $\Delta_s + m_s$ and m'_t by $\Delta_t + m_t$:

$$m(\lambda) = (1-\lambda)(m_s + \Delta_s) + \lambda(m_t + \Delta_t). \quad (2)$$

Rewriting this expression:

$$m(\lambda) = (1-\lambda)m_s + \lambda m_t + (1-\lambda)\Delta_s + \lambda\Delta_t. \quad (3)$$

Assuming $\Delta_s \approx \Delta_t$, the update term simplifies to approximately Δ_s, yielding:

$$m(\lambda) \approx (1-\lambda)m_s + \lambda m_t + \Delta_s. \quad (4)$$

or equivalently:

$$m(\lambda) \approx m_t + (1-\lambda)(m_s - m_t) + \Delta_s. \quad (5)$$

In particular, when $\lambda = 1$, $m(\lambda) = m'_t \approx m_t + \Delta_s$, which shows that reusing Δ_s corresponds to extrapolating from m_t towards the task solution learned by m_s.

Connection to Task Vector Interpolation. This interpretation aligns with prior work on task vector arithmetic (Ilharco et al., 2023), where multiple fine-tuned models are merged by adding their update vectors to a shared base. For example, the merged weights θ_m produced by adding the task vectors of model A and B (with weights θ_a and θ_b) yield:

$$\begin{aligned}\theta_m &= \theta_p + \lambda((\theta_a - \theta_p) + (\theta_b - \theta_p)) \\ &= (1-2\lambda)\theta_p + \lambda\theta_a + \lambda\theta_b\end{aligned}$$

where θ_p are the weights of the base pretrained model. This is a linear interpolation among θ_p, θ_a, and θ_b, and assumes the models lie within a connected low-loss region. Our definition of Δ_s corresponds to a special case of this framework: we apply a single update vector from m_s to a different base model m_t. Under the same connectivity assumption, this transfer remains valid and preserves task performance.

B Additional results for Section 2: Transferring fine-tuning updates across model versions

B.1 Evaluation results for Tülu and OLMo models

We also conduct experiments with Tülu (Lambert et al., 2024) and OLMo (OLMo et al., 2024), both of which were developed from Llama 3.1 through multiple alignment stages, including Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO) (Rafailov et al., 2023), and a final reinforcement learning stage—Reinforcement Learning with Verifiable Rewards (RLVR) (Lambert et al., 2024) for OLMo 2 and Tülu 3, or Group Relative Policy Optimization (GRPO) (Shao et al., 2024) for Tülu 3.1. At a high level, we subtract the weights of Llama 3.1 from these alignment-tuned checkpoints and then backport (add) the resulting diff vectors to Llama 3.0. Recycling is not applicable here, as we do not have the alignment-tuned checkpoints for Llama 3.0.

Our results are summarized in Table 7 and Table 8. In general, we find that advanced LLM capabilities, attained through alignment tuning stages such as SFT, DPO, RLVR, and GRPO (encoded in Δ_{SFT}, Δ_{DPO}, Δ_{RLVR}, and Δ_{GRPO}, respectively), can be successfully transferred across different model versions. For example, backporting Δ_{GRPO} from Tülu 3.1 8B to Llama 3.0 8B significantly improves accuracy, boosting GSM8K performance from 55.6% to 85.8% (30.2% improvement) and IFEval from 34.5% to 82.6% (48.1% improvement). This surpasses Llama 3.0 8B Instruct (81.1% on GSM8K, 76.6% on IFEval) and performs competitively with Llama 3.1 8B Instruct

Model	GSM8K	MATH	ARC$_C$	GPQA	MMLU	IFEval
Llama 3.1 8B	56.6	19.3	79.2	21.9	66.8	36.4
Llama 3.1 8B Instruct	86.5	**50.3**	**83.8**	31.3	**72.9**	80.5
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	**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	34.5
+ 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_{RLVR}	85.1	37.6	79.1	32.4	66.2	82.4
Tülu 3.1 8B	**89.9**	**43.3**	79.0	31.4	**67.6**	**84.1**
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
+ Δ_{GRPO}	85.8	39.5	78.2	29.4	65.0	82.6

Table 7: We find that advanced LLM capabilities, attained through alignment tuning stages such as SFT, DPO, RLVR, and GRPO (encoded in Δ_{SFT}, Δ_{DPO}, Δ_{RLVR}, and Δ_{GRPO}, respectively), can be successfully transferred across different model versions.

(86.5% and 80.5%) and Tülu 3.1 8B (89.9% and 84.1%).

B.2 Additional results for Section 2: Transferring fine-tuning updates across model architectures

Table 9 and Table 10 summarize fine-tuning transfer results across model versions with architectural differences. We compute the diff vector as described in Section 2, applying fine-tuning updates only to layers in the target model that match the source in shape. We observe that reusing fine-tuning updates across large version gaps remains challenging, and we leave this direction to future work.

C Additional evaluation details

We use the same evaluation setup and prompts as those in Llama 3 (Dubey et al., 2024) for Llama models and those in Tülu 3 (Lambert et al., 2024) for OLMo and Tülu models, whenever available. See Table 11 and Table 12 for more details. For evaluation, we use the `lm-evaluation-harness` library (Gao et al., 2024) for Llama models, and the `OLMES` library (Gu et al., 2024) for OLMo and Tülu models.

[8] See https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/eval_details.md

D Additional results for Section 4: When is fine-tuning transfer effective?

See Table 13.

E Additional results for Section 5: Fine-tuning transfer as a starting point for further fine-tuning

See Table 14, Figure 3.

F Additional results for Section 6: Iterative recycling-then-finetuning for improved performance and efficiency

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 models $\mathcal{M}_1^*, \mathcal{M}_2^*, \ldots, \mathcal{M}_n^*$
$\mathcal{M}_1^* \leftarrow \mathrm{FT}(\mathcal{M}_1)$
for $i = 2$ to n **do**
    $\Delta_{i-1}^{iter} = \mathcal{M}_{i-1}^* - M_{i-1}$
    $\mathcal{M}_i^* \leftarrow \mathrm{FT}(M_i + \Delta_{i-1}^{iter})$
end for
return $\mathcal{M}_1^*, \mathcal{M}_2^*, \ldots, \mathcal{M}_n^*$

Algorithm 1 provides the formal description of the iterative recycling-then-finetuning procedure.

Iterative recycling-then-finetuning leads to faster convergence: Figure 4 shows that both recycling approaches—iterative (Δ^{iter}) and direct

Model	GSM8K	MATH	ARC_C	GPQA	MMLU	IFEval
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**
$\mathcal{M}_0$	**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_{RLVR}	2.0	0.8	24.1	0.6	1.4	13.3
$\mathcal{M}_3$	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_{RLVR}	40.2	**10.3**	**75.1**	**29.9**	56.7	**68.3**
$\mathcal{M}_{4'}$	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_{RLVR}	**82.8**	27.7	**79.3**	27.2	62.2	72.1

Table 8: We find that advanced LLM capabilities, attained through alignment tuning stages such as SFT, DPO, and RLVR (encoded in Δ_{SFT}, Δ_{DPO}, and Δ_{RLVR}, respectively), can be successfully transferred across different model versions. Here, $\mathcal{M}_{4'}$ is an intermediate pretrained checkpoint of OLMo 2 7B (mid-stage 2, at 7K steps), which we selected before conducting our controlled experiments in Section 4.1.

	GSM8K	MATH
Llama 2.0 7B	14.1	3.6
+ FT	**56.9**	3.1
+ $\Delta_{3.0}$	15.0	**3.8**
+ $\Delta_{3.1}$	14.6	**3.8**
Llama 3.0 8B	54.9	17.3
+ FT	**70.7**	**32.0**
+ $\Delta_{2.0}$	55.3	17.5
Llama 3.1 8B	56.6	19.3
+ FT	**71.2**	**33.7**
+ $\Delta_{2.0}$	57.1	20.3

Table 9: Transfer results in both recycling and backporting scenarios on GSM8K and MATH show limited improvement, possibly due to layer shape mismatches.

(Δ^{dir}) recycling-then-finetuning—offer a more computationally efficient starting point for further fine-tuning. In general, Δ^{iter} consistently outperforms Δ^{dir} in terms of training efficiency and significantly improves standard fine-tuning without recycling. These results indicate that iterative recycling not only improves model performance but also enhances training efficiency by effectively leveraging the knowledge stored in the diff vectors across different model versions.

	GSM8K	MATH
OLMo 1 7B	28.8	5.8
+ FT	**54.2**	**17.2**
+ Δ_2	25.1	5.5
OLMo 2 8B	66.9	19.2
+ FT	**76.4**	**21.1**
+ Δ_1	69.7	20.1

Table 10: Fine-tuning transfer remains effective when applying Δ_1 to OLMo 2 8B on GSM8K. In other cases, improvements are limited and sometimes lead to small drops.

Task	**# Shots**	**CoT**	**Metric**	**Reference eval. setup**
GSM8K	8	✓	exact match acc.	Llama 3 Evaluation Details[8]
MATH	4	✓	exact match acc.	
ARC_C	0	✗	acc.	
GPQA	0	✓	exact match acc.	
MMLU	0	✓	exact match acc.	
IFEval	0	✗	avg. acc. (strict & loose)	
Global MMLU	0	✗	acc.	Singh et al. (2024a)
HumanEval+	0	✗	pass@1	Liu et al. (2023)
MBPP+	0	✗	pass@1	
LiveCodeBench	0	✗	pass@1	Jain et al. (2024)
BigCodeBench	0	✗	pass@1	Zhuo et al. (2024)

Table 11: Evaluation details for Llama 3.

Task	**# Shots**	**CoT**	**Metric**	**Reference eval. setup**
GSM8K	8	✓	exact match acc.	Lambert et al. (2024)
MATH	4	✓	flex exact match acc.	
ARC_C	5	✗	acc.	
GPQA	0	✓	exact match acc.	
MMLU	0	✓	exact match acc.	
IFEval	0	✗	prompt-level loose acc.	
MATH500	0	✓	exact match acc.	Muennighoff et al. (2025)
$GPQA_{Diamond}$	0	✓	exact match acc.	

Table 12: Evaluation details for OLMo 2 and Tülu 3.

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	14.6	11.6	11.4	11.6	16.6
$+ \Delta_1$		8.8	17.8	19.2	15.6
$+ \Delta_2$	7.6		12.6	14.6	14.4
$+ \Delta_3$	8.0	9.4		23.4	27.8
$+ \Delta_4$	7.8	8.0	9.8		**34.2**
$+ \Delta_5$	8.0	7.4	11.2	30.6	
FT($\mathcal{M}_i$)	13.4	**17.6**	**21.6**	**31.4**	33.0

Table 13: MATH500 accuracies indicating that more powerful models are better at leveraging transferred fine-tuning. Effective use of transferred fine-tuning only emerges once the target base model reaches a certain level of capability. Furthermore, fine-tuning transfer works best when the source and target models are close within a linearly connected region of the parameter space. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_i refers to the diff vector resulting from the fine-tuning of version i. FT($\mathcal{M}_i$) denotes applying fine-tuning directly to $\mathcal{M}_i$.

	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$
	14.6	11.6	11.4	11.6	16.6
$+ \Delta_1 \rightarrow$ FT		$21.0_{+12.2}$	$23.0_{+5.2}$	$\mathbf{32.0}_{+12.8}$	$34.2_{+18.6}$
$+ \Delta_2 \rightarrow$ FT	$16.2_{+8.6}$		$\mathbf{26.2}_{+13.6}$	$31.6_{+17.0}$	$31.0_{+16.6}$
$+ \Delta_3 \rightarrow$ FT	$\mathbf{18.4}_{+10.4}$	$21.2_{+11.8}$		$31.0_{+7.6}$	$32.0_{+4.2}$
$+ \Delta_4 \rightarrow$ FT	$17.4_{+9.6}$	$19.0_{+11.0}$	$23.8_{+14.0}$		$32.2_{-2.0}$
$+ \Delta_5 \rightarrow$ FT	$17.0_{+9.0}$	$\mathbf{21.4}_{+14.0}$	$25.0_{+13.8}$	$31.2_{+0.6}$	
FT($\mathcal{M}_i$)	13.4	17.6	21.6	31.4	33.0

Table 14: MATH500 accuracies showing that fine-tuning transfer provides a stronger starting point (i.e., $\mathcal{M}_i + \Delta_j$) for further fine-tuning (FT). Numbers in subscript indicate improvement over the baseline without fine-tuning. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_i refers to the diff vector resulting from the fine-tuning of version i. FT($\mathcal{M}_i$) denotes applying fine-tuning directly to $\mathcal{M}_i$.

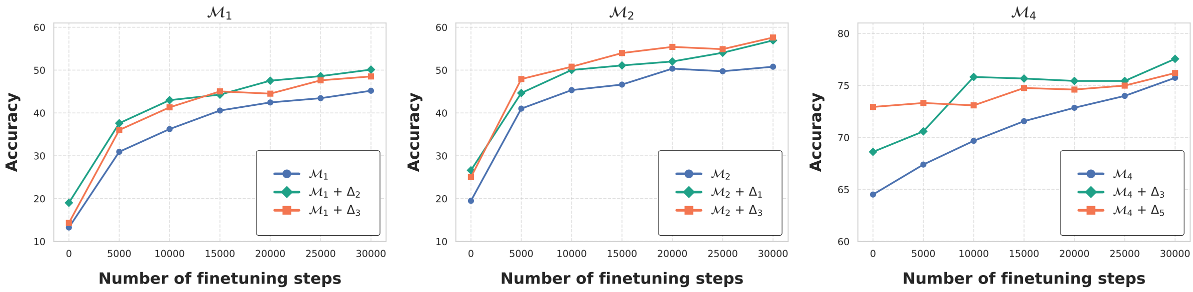

Figure 3: GSM8K performance showing that fine-tuning transfer provides a more computationally efficient starting point (i.e., $\mathcal{M}_i + \Delta_j$) for further training. Here, $\mathcal{M}_i$ represents different intermediate pretrained checkpoints of OLMo 2 7B (with smaller values of i indicating earlier checkpoints), and Δ_i refers to the diff vector resulting from the fine-tuning of version i.

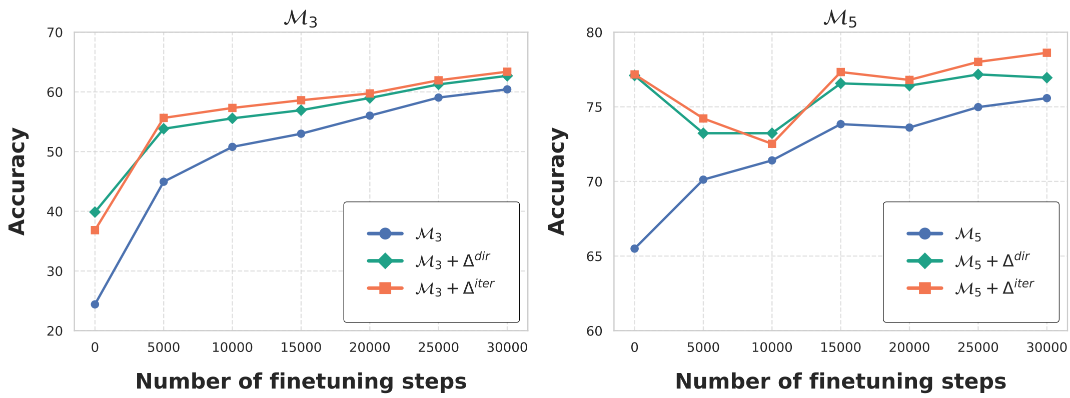

Figure 4: GSM8K performance showing that both iterative (Δ^{iter}) and direct (Δ^{dir}) recycling-then-finetuning approaches offer faster convergence. At a high level, Δ^{iter} gradually incorporates fine-tuning updates, i.e., diff vectors, from previous model versions, while Δ^{dir} directly applies the diff vector from the latest model version to the current model.