# OPTIMIZER-MODEL CONSISTENCY: FULL FINETUNING WITH THE SAME OPTIMIZER AS PRETRAINING FORGETS LESS

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

## ABSTRACT

Optimizers play an important role in both pretraining and finetuning stages when training large language models (LLMs). In this paper, we present an observation that full finetuning with the same optimizer as in pretraining achieves a better learning-forgetting tradeoff, i.e., forgetting less while achieving the same or better performance on the new task, than other optimizers and, possibly surprisingly, LoRA, during the supervised finetuning (SFT) stage. We term this phenomenon optimizer-model consistency. To better understand it, through controlled experiments and theoretical analysis, we show that: 1) optimizers can shape the models by having regularization effects on the activations, leading to different landscapes around the pretrained checkpoints; 2) in response to this regularization effect, the weight update in SFT should follow some specific structures to lower forgetting of the knowledge learned in pretraining, which can be obtained by using the same optimizer. Moreover, we specifically compare Muon and AdamW when they are employed throughout the pretraining and SFT stages and find that Muon can perform worse when finetuned for reasoning tasks. With a synthetic language modeling experiment, we provide some evidence that this can come from Muon's strong tendency towards rote memorization, which may hurt pattern acquisition with a small amount of data, as for SFT.

## 1 INTRODUCTION

Large language models (LLMs) have achieved remarkable success under the pretrain-finetune training paradigm (Radford et al., 2019; Ouyang et al., 2022; Grattafiori et al., 2024; Yang et al., 2025). In the pretraining stage, the models are trained on an enormous mixed corpus to obtain general language ability. With the pretrained checkpoint in hand, we typically start the finetuning stage with supervised finetuning (SFT), where models are trained upon the pretrained checkpoints with high-quality but relatively small amounts of data to learn the skills in specific downstream tasks, e.g., math and code. SFT is a significantly different training stage from pretraining, since models not only need to learn novel knowledge and skills, but also preserve the general language ability learned in pretraining. We call balancing the two factors the learning-forgetting tradeoff.

Training algorithms, or optimizers, play a central role in the learning-forgetting tradeoff. Among the optimizers, the current workhorse for LLM training is undoubtedly AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017) and its variants, with Muon (Jordan et al., 2024) and other recently emerged matrix-structured optimizers as strong competitors applied in frontier model training (Kimi et al., 2026; Zeng et al., 2025; DeepSeek-AI, 2026). Specifically in SFT, parameter-efficient training methods are also powerful, with LoRA (Hu et al., 2022) as the most popular and representative one. The rich diversity in training algorithms makes us wonder: *Which training algorithms achieve the best learning-forgetting tradeoff in the SFT stage?*

In this paper, we provide an answer to the question: full finetuning with the same (family of) optimizer as pretraining generally achieves a better learning-forgetting tradeoff compared to other optimizers and LoRA. As presented in Figure 1, we plot a Pareto frontier where each point corresponds to a learning rate choice and a training period for a specific training method. The matching optimizer—AdamW for Llama-2-7B (left) and Muon for RNJ-1 (right)—generally gives a bet-



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

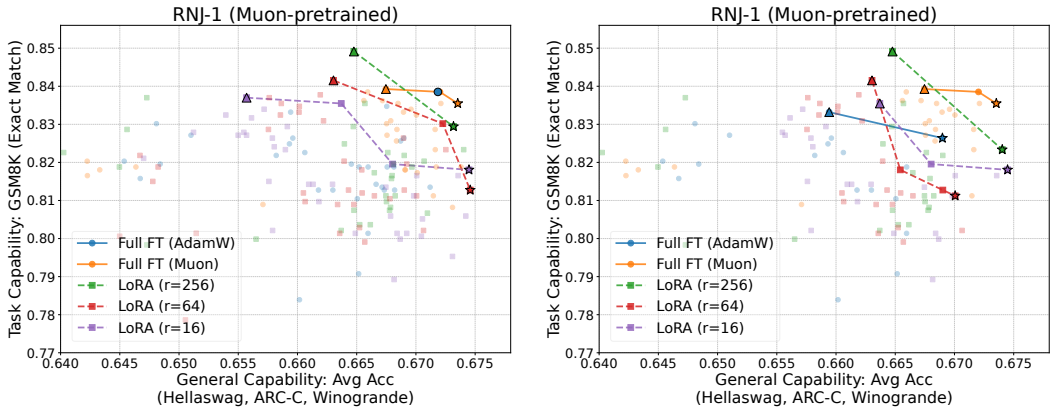



Figure 1: Pareto frontiers for finetuning Llama-2-7B (left) and RNJ-1 (right) on MetaMathQA. Full finetuning with the pretraining optimizer generally achieves a better learning-forgetting tradeoff. LoRA uses the matching optimizer. Higher is better on both axes.

ter learning-forgetting tradeoff among full finetuning methods, and also outperforms LoRA. The same phenomenon is also observed across our controlled GPT-2 experiments., which we term it **optimizer-model consistency**. Note that related observations have appeared in work on optimizer mismatch (Liu et al., 2025) and LoRA forgetting (Biderman et al., 2024; Shuttleworth et al., 2024). Concurrent work also studies how learning rates affect forgetting (Rofin et al., 2026) and how LoRA mitigates optimizer mismatch (Qu et al., 2026). Our focus is not only on presenting the comparison between optimizers in full finetuning, but also studying learning-forgetting frontiers across learning rates and checkpoints, showing that full finetuning can potentially achieve better training-forgetting tradeoff compared to LoRA with the matching optimizer, and providing explanations and understanding of this phenomenon through how pretraining optimizers affect SFT training.

To understand why optimizer-model consistency occurs, we start by considering the difference in the models pretrained by different optimizers. Since LLM training is generally nonconvex, different (families of) optimizers consistently favor solutions with specific properties. We discuss the properties in weights and activations, and consider the optimizer framework presented in Pethick et al. (2025) with matrix induced norms, showing that the activation properties of the model can originate from the optimizer's regularization effect. Then, based on the understanding of the activation regularization effect, we provide a theoretical explanation for optimizer-model consistency under specific assumptions modeling the observations.

From the understanding of optimizer-model consistency, we know that we should employ the same (family of) optimizers in SFT to have a better learning-forgetting tradeoff. It also implies that the learning-forgetting tradeoff when an optimizer is employed throughout pretraining and SFT is a comprehensive metric for an optimizer in LLM training compared to only pretraining performance. We conduct a case study for Muon and AdamW in this way and observe that though Muon generally provides a stronger pretrained checkpoint, its performance after SFT varies across different tasks, and can possibly be no better than AdamW if finetuned on reasoning tasks like math. We further conduct a synthetic experiment to suggest that a possible reason is that Muon may be better at memorization and potentially weaker at learning patterns due to its update formula, which is particularly significant in tasks with only a small amount of data, like SFT. Therefore, a better finetuning method may be an interesting direction for improving Muon's overall performance in LLM training.

We summarize our contributions:

- We demonstrate the **optimizer-model consistency** phenomenon: full finetuning with the same optimizer as pretraining leads to a better learning-forgetting tradeoff than full finetuning with other optimizers or LoRA in the SFT stage. Specifically, full finetuning can surpass LoRA in learning-forgetting tradeoff regardless of the LoRA optimizer when a learning rate sweep is properly considered (Section 2).

- We show that an optimizer generally applies regularization to the activations related to the norm on which it is derived. We build a theoretical explanation for the optimizer-

model consistency phenomenon under simplified assumptions by connecting the activation properties and how SFT update affects the forgetting of pretraining knowledge (Section 3).

- We specifically compare Muon and AdamW when they are employed throughout pretraining and SFT (Section 4) and find that Muon can lose its advantages when finetuning on reasoning tasks like math. We provide evidence for a possible explanation that this may be because Muon tends to memorize rather than learn patterns through a synthetic language modeling experiment (Section 4).

Due to the page limit, we defer the related work section to Appendix A.

## 2 Full Finetuning with the Same Optimizer Forgets Less

In this section, we present the optimizer-model consistency phenomenon using GPT-2-small (Ouyang et al., 2022), Llama-2-7B (Touvron et al., 2023), and the Muon-pretrained RNJ-1 model. To better compare the SFT performance across different optimizers under different checkpoints, we conduct a full pretrain-finetune training pipeline on the GPT-2 model. We employ OpenWebText (Gokaslan & Cohen, 2019) as the pretraining dataset, and consider three different types of tasks for SFT: MetaMathQA (Yu et al., 2023) for math, Alpaca (Taori et al., 2023) for instruction following, Magicoder-Evol-Instruct-110K (Wei et al., 2023) for coding. For GPT-2 experiments, we evaluate the OpenWebText validation loss as a measure for forgetting, i.e., how much the models preserve the pretraining knowledge, and the loss on the new task validation loss as a measure for learning, i.e., how much the models learn in SFT (lower is better). For a comprehensive and fair comparison between algorithms, we plot the Pareto frontier of the learning-forgetting tradeoff, with $x$ and $y$-axis denoting the forgetting and learning metric, respectively. For each training algorithm, we consider all the runs in a learning rate grid search, which always contains the learning rate that achieves the best SFT performance, and we evaluate intermediate checkpoints for a single run. Then, we gather all the data points for each training algorithm to present the Pareto frontier to find the limit of this training algorithm in learning and forgetting. The detailed experiment settings can be found in Appendix B.1.

**Muon achieves better pretraining performance compared to AdamW.** We present the results of the models pretrained by Adam and Muon in Table 1. We can see that Muon outperforms AdamW across the pretraining dataset and different downstream tasks, validating the favorable performance of Muon in pretraining (Jordan et al., 2024; Liu et al., 2025; Wen et al., 2025b).

Table 1: The validation loss of different datasets after pretraining on GPT-2.

| Optimizer for Pretraining | OpenWebText | Alpaca | MetaMathQA | Magicoder |
|---|---|---|---|---|
| AdamW | 3.010 | 3.483 | 3.510 | 3.264 |
| Muon | 2.974 | 3.235 | 3.200 | 3.000 |

**The same optimizer as pretraining leads to the best learning-forgetting tradeoff.** With the pretrained checkpoints in hand, we start the SFT stage. The Pareto frontier plots of full finetuning with AdamW and Muon to quantify the learning-forgetting tradeoff across different downstream tasks are summarized in Figure 2, where the $x$-axis denotes the OpenWebText validation loss and the $y$-axis denotes the task validation loss. One can observe that the same optimizer as pretraining always yields the best tradeoff, no matter whether it is AdamW or Muon, which is consistent with the observation in Liu et al. (2025) on an instruction following task. It is also worth noting that SGD achieves a reasonable downstream performance, indicating its capability to learn from a small amount of data, but it forgets significantly more when the same downstream ability is learned.

**Full finetuning can forget less than LoRA with appropriate hyperparameters.** We also evaluate the learning-forgetting tradeoff of LoRA, which is commonly believed to forget less because of its low-rank nature (Biderman et al., 2024). The results are presented in Figure 3. We observe that LoRA generally tends to forget less when it achieves the best learning performance (the best $y$-axis value), which is consistent with the common intuition and existing results. However, when taking a full view of the tradeoff, full finetuning can achieve less forgetting at the same or better learning over much of the evaluated frontier. Therefore, LoRA in this setting generally gives a worse learning-forgetting tradeoff compared to full finetuning with the same optimizer, and additionally, we can observe that a larger LoRA rank generally corresponds to a better tradeoff. While LoRA is an

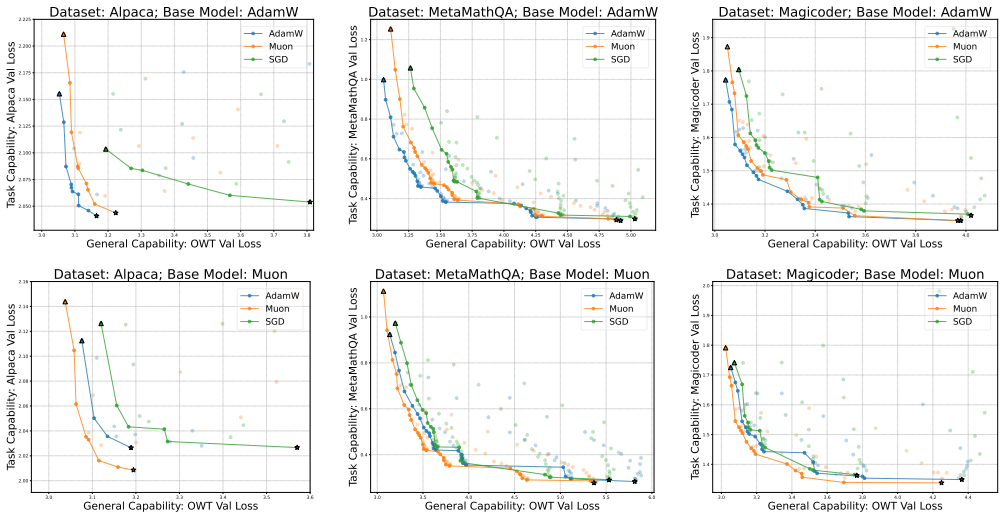



Figure 2: Pareto frontier plots of full finetuning with AdamW and Muon optimizers on GPT-2. The rows and columns present the results of different pretraining optimizers and SFT datasets, respectively. The measure for learning and forgetting is the corresponding validation loss (lower is better). We can observe that the same optimizer performs better in the forgetting-learning trade-off in SFT.

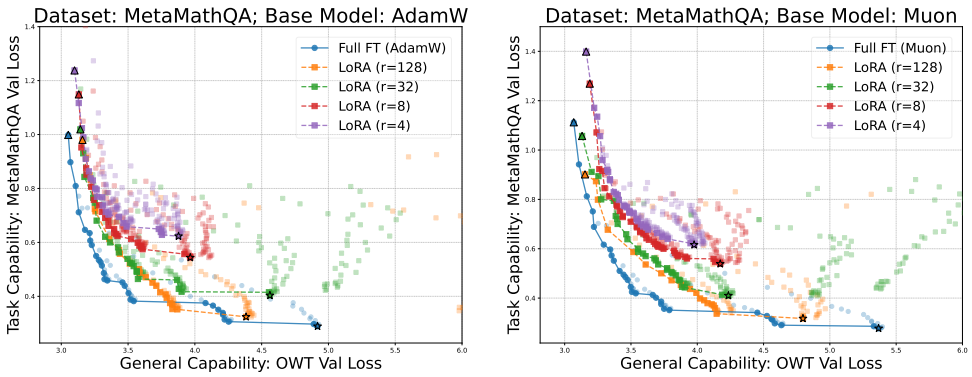



Figure 3: Pareto frontiers of full finetuning and LoRA on GPT-2 with the same optimizer as pretraining employed for both. Full finetuning generally achieves a better learning-forgetting tradeoff.

effective training algorithm with incredible parameter efficiency, the results suggest that it does not necessarily help in improving learning-forgetting tradeoff performance. Additional MetaMathQA runs with two different seeds are reported in Appendix C.3.

**Results on larger models.** As shown in Figure 1, our conclusions in small models hold still when finetuning Llama-2-7B with MetaMathQA, generally following the settings of Biderman et al. (2024). We can observe that full finetuning with Adam (the same optimizer as Llama-2-7B pretraining) yields a better learning-forgetting tradeoff, even without weight decay, compared to other full finetuning optimizers and LoRA. On Muon-pretrained RNJ-1 (8.3B parameters), full Muon generally achieves a better learning-forgetting tradeoff than full AdamW, while Muon-trained LoRA can reach a higher peak accuracy. See Appendix C.1 for experiment settings.

For brevity, we name this phenomenon that full finetuning with the same optimizer as pretraining leads to the best learning-forgetting tradeoff as **optimizer-model consistency**.

**Remark 1** (Discussion on Results in Biderman et al. (2024))**.** *Our results are compatible with Biderman et al. (2024), whose main tradeoff curves vary training duration after selecting learning rates and already include settings favoring full finetuning. We also observe that LoRA learns less and forgets less when selecting learning rates for the best task performance (Figure 8). Jointly considering learning rates and checkpoints can reveal full finetuning settings with both less forgetting and better learning. The role of SFT learning rates is also discussed in (Rofin et al., 2026) concurrently.*

# 3 WHY OPTIMIZER-MODEL CONSISTENCY HAPPENS

The optimizer-model consistency phenomenon may seem surprising at first glance if we consider optimizers or training algorithms only as processes towards some fixed solutions. However, optimizers actually have a much larger effect on the trained model than this common impression. In LLM training or other deep learning tasks, the losses are generally nonconvex, and different optimizers can lead to very different trajectories and convergence points. In LLM training or other deep learning tasks, the losses are generally nonconvex, and different optimizers can consistently lead to significantly different trajectories and convergence checkpoints. The difference in checkpoints is more than random noise, but consistent properties in the weights and activations of the model, which we will discuss in Section 3.1. Then, with these properties, we have the basis to better understand why optimizer-model consistency happens in Section 3.2.

## 3.1 OPTIMIZERS SHAPE MODELS

**Optimizers shape weights.** We start with the observations on the difference in model weights trained by different optimizers through pretraining. As presented in Table 2, we can observe that the model trained by Muon shows a denser weight spectrum distribution (higher stable rank and singular sparsity value), which is consistent with the observations in Liu et al. (2025); Wang et al. (2025). A straightforward intuition for this spectrum difference is that Muon updates are always approximately full-rank because of its formula, resulting in the final weights, an accumulation of all the updates, being closer to full-rank compared to AdamW.

Table 2: Singular spectrum comparison after pretraining. For a singular spectrum $\sigma_1 \geq \cdots \geq \sigma_n$, the stable rank is $\sum_{i=1}^{n} \sigma_i^2/\sigma_1^2$ and singular sparsity is $\sum_{i=1}^{n} \sigma_i/\sqrt{n\sum_{i=1}^{n} \sigma_i^2}$. We take the average of the values over all linear layers in GPT-2.

| Optimizer | Stable Rank | Singular Sparsity |
|---|---|---|
| AdamW | 65.75 | 0.837 |
| Muon | 114.72 | 0.875 |

**Optimizers regularize activations.** In addition to explicit weight properties, optimizers also affect the activations of trained models on the pretraining dataset, which is closely related to how the models store pretraining knowledge. As presented in Figure 4, we can observe that the activations, especially input activations, of models trained by AdamW are much sparser compared to those by Muon.

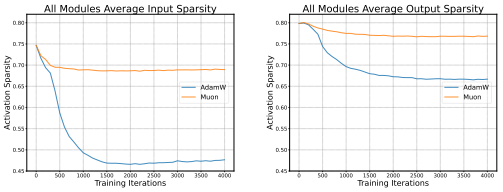



Figure 4: Activation sparsity of AdamW and Muon. We adopt the same pretraining settings as Section 2, but for 4k iterations. The $y$-axis represents the sparsity of activations computed by $\|x\|_1 / (\sqrt{d}\,\|x\|_2)$ for an activation vector $x$ (lower means sparser). We take an average of the input and output sparsity over all the linear layers.

To understand how this difference in activations originates, we start from the following framework for optimizer derivation (Pethick et al., 2025): with $G_t$ as the gradient of weight $W_t \in \mathbb{R}^{m\times n}$, we update the weight by

$$W_{t+1} = \operatorname*{argmin}_{\|W-W_t\|\leq R} \langle G_t, W - W_t \rangle, \tag{1}$$

where we consider a specific norm $\|\cdot\|$. Since the weights of neural nets generally have a matrix structure and the activation computation is by matrix-vector multiplication, we focus on the matrix induced norms under the framework: for $\alpha, \beta \in [1, \infty]$, we denote $\|A\|_{\alpha,\beta} \triangleq \max_{x\neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}$ and the optimizer derived from $\|\cdot\|_{\alpha,\beta}$ as $\mathcal{A}_{\alpha,\beta}$. Note that Muon is basically $\mathcal{A}_{2,2}$ from its derivation (Jordan et al., 2024; Bernstein & Newhouse, 2024) and SignSGD (Bernstein et al., 2018) is $\mathcal{A}_{1,\infty}$ by noticing that the vector infinite norm is equivalent to $\|\cdot\|_{1,\infty}$. Following Balles & Hennig (2018); Balles et al. (2020); Kunstner et al. (2023), we consider Adam as a powerful variant of SignSGD, thus it also falls in the family of $\mathcal{A}_{1,\infty}$. Pethick et al. (2025); Xu et al. (2026) discuss additional optimizers derived from this framework with matrix induced norms, demonstrating its generality.

For an optimizer $\mathcal{A}_{\alpha,\beta}$, the weight $W$ is updated with $\|\cdot\|_{\alpha,\beta}$ bounded. Besides a direct constraint on the weights, this can also be considered as some forms of regularization for the activations:

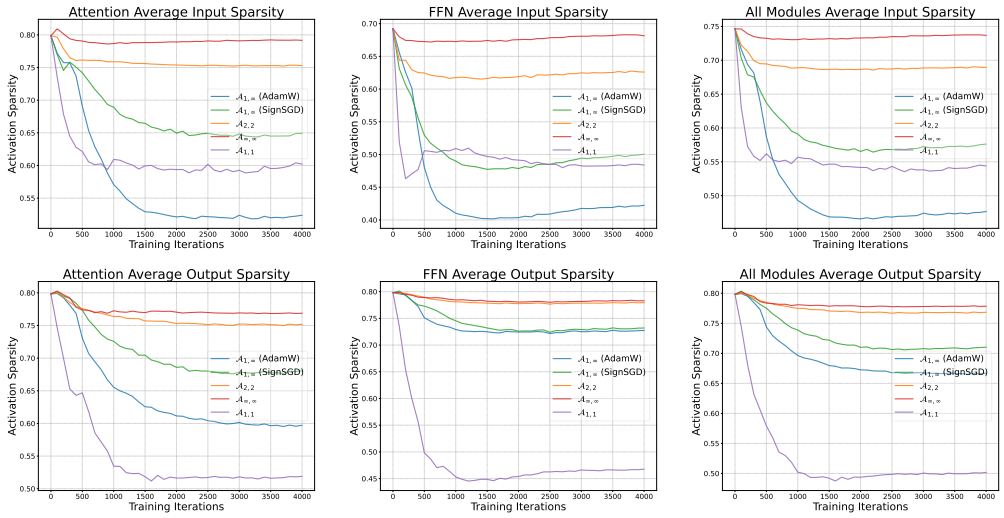



Figure 5: Activation sparsity of different optimizers $\mathcal{A}_{\alpha,\beta}$ derived from (1) with the same settings as Figure 4. The columns present the average sparsity of different modules, and the columns present input and output activations of the modules. We only consider the linear layers for doing this average.

regularizing the $\|\cdot\|_\alpha$ input activation $x$ and $\|\cdot\|_\beta$ for the output activation $y$, since we have $y = Wx$ and the matrix induced norm definition $\|y\|_\beta \leq \|W\|_{\alpha,\beta} \|x\|_\alpha$. Specifically, we have:

- Muon is $\mathcal{A}_{2,2}$, thus regularizes input $x$ by $\|\cdot\|_2$ and output $y$ by $\|\cdot\|_2$.
- Adam is $\mathcal{A}_{1,\infty}$, thus regularizes input $x$ by $\|\cdot\|_1$ and output $y$ by $\|\cdot\|_\infty$.

Different regularizations lead to different properties of activations. For example, $\|\cdot\|_1$ regularization corresponds to Lasso, which is widely known to result in a sparse solution. As presented in Figure 5, empirical observations verify our intuition about activation properties. In addition to SignSGD ($\mathcal{A}_{1,\infty}$), AdamW($\mathcal{A}_{1,\infty}$), and Muon ($\mathcal{A}_{2,2}$), we also consider algorithms $\mathcal{A}_{1,1}$ and $\mathcal{A}_{\infty,\infty}$ for a more comprehensive view of the regularization effect. Their details can be found in Appendix B. We can observe that for an algorithm $\mathcal{A}_{\alpha,\beta}$, the input activation is consistently sparser when $\alpha$ is taken to be 1, which holds for $\mathcal{A}_{1,\infty}$ and $\mathcal{A}_{1,1}$. The same argument also holds for the output activations when $\beta = 1$, which verifies the regularization effect of activations and its relation to the optimizer choice.

### 3.2 The SFT Optimizer should Fit the Activation Regularization

Based on the observations that optimizers can shape models, we provide a theoretical explanation of the optimizer-model consistency phenomenon under a simplified setting. The intuition is straightforward: since different optimizers lead to pretrained checkpoints with different properties, forgetting gets less when the SFT update better matches the curvature around the checkpoints, which can be achieved with full finetuning using the same optimizer. We start with approximations of the forgetting and SFT loss, based on the fact that only a small weight change is applied in the SFT stage.

**Approximating forgetting.** We have the following Taylor expansion approximation of the pretraining loss increase after SFT:

$$\begin{aligned}\mathcal{L}_{\text{pre}}(\theta_0 + \Delta\theta) \approx& \mathcal{L}_{\text{pre}}(\theta_0) + \langle \nabla_\theta \mathcal{L}_{\text{pre}}(\theta_0), \Delta\theta \rangle + \frac{1}{2} \Delta\theta^\top \nabla_\theta^2 \mathcal{L}_{\text{pre}}(\theta_0) \Delta\theta \\ \approx& \mathcal{L}_{\text{pre}}(\theta_0) + \frac{1}{2} \Delta\theta^\top \nabla_\theta^2 \mathcal{L}_{\text{pre}}(\theta_0) \Delta\theta,\end{aligned}$$

where we denote $\mathcal{L}_{\text{pre}}$ as the pretraining loss, $\theta_0$ as the model weight after pretraining, and $\Delta\theta$ as the weight change after SFT, whose scale should be small. The second line holds since the model should approximately arrive at a local minima after pretraining, which yields that $\nabla_\theta \mathcal{L}_{\text{pre}}(\theta_0) \approx 0$.

Based on the observations that the Hessians of neural networks are typically block-wise diagonal with respect to layers (Martens & Grosse, 2015; Singh et al., 2021; Nayak et al., 2025), we then consider the forgetting loss in a layer-wise manner without loss of generality, since the total loss can be obtained by summing up all the layers. For a linear layer $W \in \mathbb{R}^{m \times n}$ of the model $\theta$, we define

$$\hat{\mathcal{L}}_{\text{forget}}(\Delta W) \triangleq \frac{1}{2} \text{vec}\left(\Delta W\right)^\top \nabla_W^2 \mathcal{L}_{\text{pre}}(\theta_0) \text{vec}\left(\Delta W\right),$$

where $\Delta W$ is the update of the layer $W$ after SFT and the $\mathrm{vec}(W)$ operation means vectorizing $W$ by stacking the rows, i.e., $\mathrm{vec}(W) \triangleq [w_1, \ldots, w_m] \in \mathbb{R}^{mn\times 1}$ for $W = [w_1^\top; \ldots; w_m^\top] \in \mathbb{R}^{m\times n}$. We have the input and output activations of the layer $W$ to be $x, y$ such that $y = Wx$ when the pretraining data is fed to the model. We further consider the Fisher Information matrix $\mathbb{E}[gg^\top]$ as an effective approximation of the Hessian $\nabla^2_W \mathcal{L}_{\mathrm{pre}}(\theta_0)$ (Martens & Grosse, 2015), where $g = \mathrm{vec}(\nabla_W \mathcal{L}_{\mathrm{pre}}(\theta_0))$ is the vectorized gradient of layer $W$ at $\theta_0$. Based on the chain rule, it holds that $\nabla_W \mathcal{L}_{\mathrm{pre}}(\theta_0) = \delta x^\top$ with $\delta \triangleq \nabla_y \mathcal{L}_{\mathrm{pre}}(\theta_0)$. Then, by substituting this gradient form, we have

$$\begin{aligned}\nabla^2_W \mathcal{L}_{\mathrm{pre}}(\theta_0) \approx& \mathbb{E}[gg^\top] = \mathbb{E}[\mathrm{vec}\left(\delta x^\top\right) \mathrm{vec}\left(\delta x^\top\right)^\top] = \mathbb{E}[(\delta \otimes x)(x \otimes \delta)^\top] \\ =& \mathbb{E}[(\delta\delta^\top) \otimes (xx^\top)] \approx \mathbb{E}[\delta\delta^\top] \otimes \mathbb{E}[xx^\top],\end{aligned}$$

where $\otimes$ denotes the Kronecker product, and the expectation is taken over the input $x$. The second approximation applies the assumption in Martens & Grosse (2015) that we can split the expectation for input activations and output gradients. Based on the observations that the Hessians are commonly block-wise diagonal with respect to rows of MLPs (Collobert, 2004; Zhang et al., 2024a;b), we further simplify the Hessian approximation to be mostly relevant to the input activation expectation, i.e., $\nabla^2_W \mathcal{L}_{\mathrm{pre}}(\theta_0) \propto \mathbb{E}[xx^\top]$. Then, we can approximate the forgetting loss by

$$\begin{aligned}\hat{\mathcal{L}}_{\mathrm{forget}}(\Delta W) \triangleq& \frac{1}{2}\mathrm{vec}(\Delta W)^\top \nabla^2_W \mathcal{L}_{\mathrm{pre}}(\theta_0)\mathrm{vec}(\Delta W) \\ \propto& \frac{1}{2}\mathrm{tr}\left(\Delta W \mathbb{E}[xx^\top] \Delta W^\top\right) = \frac{1}{2}\mathbb{E}\left[\|\Delta W x\|_2^2\right],\end{aligned}$$

where the last equality is based on the property of trace. Therefore, we take Assumption 1.

**Assumption 1** (Approximation of forgetting loss increase)**.** *Consider the SFT update $\Delta W \in \mathbb{R}^{m\times n}$. The increase in loss caused by SFT can be approximated by*

$$\mathcal{L}_{\mathrm{forget}}(\Delta W) \triangleq \frac{1}{2}\mathbb{E}\left[\|\Delta W x\|_2^2\right], \tag{2}$$

*where $x$ denotes the input activation of pretraining data at $\theta_0$ and the expectation is taken over $x$.*

Moreover, to better quantify the multiplication of weights and activations, we consider their alignment and propose Assumption 2. Note that this alignment condition with $\alpha = \beta = 2$ between weight and the corresponding activations in the pretraining stage has been employed and examined in Yang et al. (2023). We here consider a more general version with $\alpha, \beta$, and since we always have $\|Wx\|_\beta \le \|W\|_{\alpha,\beta} \|x\|_\alpha$ based on the definition of $\|\cdot\|_{\alpha,\beta}$, Assumption 2 generally implies that the best pair $\alpha, \beta$ can achieve this lower bound. Throughout the paper, we use $\Theta(\cdot)$ and $\mathcal{O}(\cdot)$ notation to omit absolute constants not related to dimensionality and the variable scales.

**Assumption 2** (Weight activation alignment)**.** *Consider the SFT update $\Delta W \in \mathbb{R}^{m\times n}$ and the input activation of pretraining data $x \in \mathbb{R}^n$ at $\theta_0$. Fixing $\beta_* \in [1, \infty]$, there exist $\alpha_* \in [1, \infty]$ such that*

$$\mathbb{E}\left[\|\Delta W x\|_{\beta_*}^2\right] = \Theta\left(\|\Delta W\|_{\alpha_*,\beta_*}^2 \, \mathbb{E}\left[\|x\|_{\alpha_*}^2\right]\right).$$

**Approximating SFT loss.** Since the model moves only a small distance during SFT, we consider the following first-order Taylor expansion as an approximation of the SFT loss.

**Assumption 3** (SFT loss approximation)**.** *We approximate the SFT loss by*

$$\mathcal{L}_{\mathrm{SFT}}(\Delta W) \triangleq H_0 + \langle \Delta W, G \rangle,$$

*where $G \in \mathbb{R}^{m\times n}$ is the gradient of the SFT loss at $\theta_0$ and $H_0$ is the SFT loss at $\theta_0$.*

**Optimizer class.** We consider the optimizer class following framework (1) with $\|\cdot\|_{\alpha,\beta}$, including Adam and Muon. We formalize the observations on the activation regularization effects of optimizers observed in Figures 4 and 5 as in Assumption 4. While this assumption may be slightly strong for the activations, it provides a straightforward alignment with the intuition that, when a vector is regularized by $\|\cdot\|_p$, it will be shaped to minimize $\|\cdot\|_p$ while other norms can be larger.

**Assumption 4** (Optimizers regularize activations)**.** *We assume the pretraining and SFT optimizers are chosen based on the framework (1). For pretraining optimizer $\mathcal{A}_{\alpha_1,\beta_1}$ and SFT optimizer $\mathcal{A}_{\alpha_2,\beta_2}$ and the SFT update $\Delta W$ derived by $\mathcal{A}_{\alpha_2,\beta_2}$ and $y \triangleq \Delta W x$, it holds that for all $\alpha, \beta \in [1, \infty]$,*

$$\mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \begin{cases} \Theta\left(\mathbb{E}\left[\|x\|_\alpha^2\right]\right), & \alpha \ge \alpha_1, \\ \Theta\left(n^{\frac{1}{\alpha_1}-\frac{1}{\alpha}} \mathbb{E}\left[\|x\|_\alpha^2\right]\right), & \alpha < \alpha_1, \end{cases} \quad \mathbb{E}\left[\|y\|_{\beta_2}^2\right] = \begin{cases} \Theta\left(\mathbb{E}\left[\|y\|_\beta^2\right]\right), & \beta \ge \beta_2, \\ \Theta\left(m^{\frac{1}{\beta_2}-\frac{1}{\beta}} \mathbb{E}\left[\|y\|_\beta^2\right]\right), & \beta < \beta_2. \end{cases}$$

Under these assumptions, we show that full finetuning with the same family of optimizer outperforms other optimizer choices in Theorem 1 for this simplified problem scenario.

**Theorem 1** (Same (family of) optimizer forgets less)**.** *Under Assumption 1-4, we can achieve (near) optimal learning-forgetting tradeoff when taking $\alpha_2 = \alpha_1$ and $\beta_2 \in [2, \infty]$ for the SFT optimizer $\mathcal{A}_{\alpha_2,\beta_2}$, i.e., the least forgetting while achieving the same SFT performance. Formally, for any constant $C$ and $\Delta W(\mathcal{A}_{\alpha,\beta})$ such that $\mathcal{L}_{\text{SFT}}(\Delta W(\mathcal{A}_{\alpha,\beta})) \leq C$, we have*

$$\mathcal{L}_{\text{forget}}\left(\Delta W(\mathcal{A}_{\alpha_2,\beta_2})\right) = \mathcal{O}\left(\inf_{\alpha\in[1,\infty],\beta\in[1,2]} \mathcal{L}_{\text{forget}}\left(\Delta W(\mathcal{A}_{\alpha,\beta})\right)\right)$$

*with $\alpha_2 = \alpha_1$ and $\beta_2 \geq 2$, where $\Delta W(\mathcal{A}_{\alpha,\beta})$ denotes the SFT update produced by $\mathcal{A}_{\alpha,\beta}$.*

**Remark 2** (A suggestion for the training choice of SFT)**.** *We show that full finetuning with optimizers in the same family as the pretraining optimizer (or more concretely, with the same $\alpha$ and $\beta \geq 2$) obtains the best learning-forgetting tradeoff. For instance, we may consider Adam, SignSGD (Bernstein et al., 2018), or Lion (Chen et al., 2023) and their variants for finetuning AdamW checkpoints. Also note that LoRA generally fails to achieve the best tradeoff under the settings, since its adapter structure cannot fit the optimal choice for activations regularized by any norm, as long as we pretrain the model with an optimizer derived from (1), which includes Adam, Muon, and their variants.*

**SignSGD supports the optimizer-family prediction.** SignSGD with momentum achieves a tradeoff comparable to AdamW on AdamW-pretrained GPT-2 (Appendix C.2), consistent with their shared $\mathcal{A}_{1,\infty}$ family and the prediction of Theorem 1.

## 4 A Case Study on Muon Applied to Pretrain and SFT

Given that we should employ the same (family of) optimizer in SFT as in pretraining to obtain the best performance because of optimizer-model consistency, the learning-forgetting tradeoff when an optimizer is employed throughout the pretrain and SFT stages is a valid and comprehensive metric for the optimizer in training LLMs, since finetuning is usually needed to improve the model. In this section, we provide a case study to compare Muon with AdamW in this way to examine its potential in LLM training. We employ the same settings as presented in Section 2 for GPT-2.

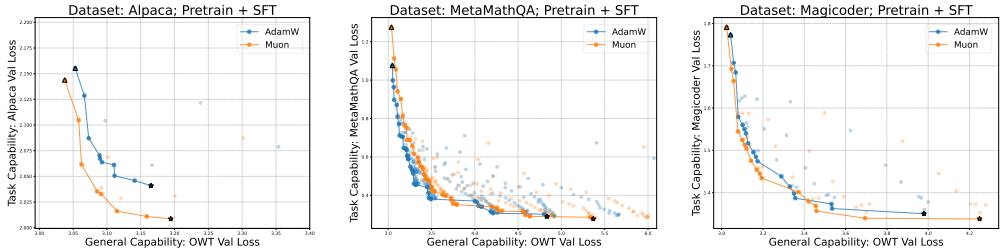



Figure 6: The comparison between Muon and AdamW when they are employed throughout the pretraining and SFT stages in Alpaca, MetaMathQA, and Magicoder tasks.

When Muon and AdamW are employed throughout the pretraining and SFT stages, the learning-forgetting tradeoff is presented in Figure 6. We can clearly observe that Muon achieves a significantly better learning-forgetting tradeoff when finetuned on Alpaca than AdamW. On math and code tasks, however, Muon is generally comparable to AdamW: slightly better in Magicoder, and slightly worse in MetaMathQA. As demonstrated in Table 1, Muon performs exceptionally better compared to AdamW before SFT in all three SFT tasks and the pretraining task. Therefore, the results in Figure 6 generally imply a degradation in the performance of Muon after finetuning on code and math tasks, showing a potentially worse capability of learning reasoning ability in SFT with Muon.

### 4.1 Muon Memorizes More, but May Learn Less

We hypothesize that *Muon's worse performance on finetuning reasoning tasks may be because Muon tends to memorize before patterns in the data have been fully acquired*, which can hurt learning when only a small amount of data is available, as in the SFT stage. This is consistent with Muon's performance variation in SFT, since reasoning tasks generally require learning new abilities, while

instruction following tasks are more related to memorization. Our hypothesis originates from the fact that each update of Muon is full-rank, potentially being able to memorize more knowledge compared to the typically lower-rank updates of AdamW and SGD (Yang et al., 2023), suggesting that rote memorizing data is easier for Muon. When memorizing is easier, models can tend to do rote memorization instead of learning patterns to lower loss faster, leading to worse generalization and undesirable performance.

To provide evidence for our hypothesis, we conduct a synthetic experiment on a corrupted language corpus, in which we split the data into clean and corrupted sequences. The corrupted sequences are generated from the original clean sequences by randomly shuffling the order of tokens, e.g., for a complete sentence "*attention is all you need*", a random shuffle can be "*is need you all attention*". In this way, we destroy the internal patterns of language in corrupted sequences, so that only rote memorization is possible, similar to the randomization test (Edgington & Onghena, 2007) applied in Zhang et al. (2016). By mixing clean and corrupted sequences, we can assess the optimizers' tendency to memorize and learn patterns by checking how well they learn on clean and corrupted data separately. The detailed experiment settings and the evaluation metric of exact match are in Appendix B.2.

**Models learn patterns first.** The memorization accuracy vs iteration plot is presented in Figure 12. We can observe that with all three optimizers, models memorize the clean data much faster than the corrupted random data, which generally implies the existence of patterns in natural language, making it easier to learn. Also, the models start to learn corrupted data (by rote memorization) only after having some level of capacity for the clean data, which is consistent with the observation that models learn simple patterns before memorization in computer vision tasks (Arpit et al., 2017).

**Muon tends to memorize before fully acquiring the patterns.** As we can observe from Table 3 and Figure 12, Muon achieves higher memorization accuracy for the corrupted data, but lower for clean data, compared to AdamW and SGD. Also, the corrupted data memorization accuracy of Muon starts to increase earlier than the other two optimizers, showing a tendency to start rote memorization earlier. Also note that the train loss of Muon is still the lowest, showing the fastest speed in lowering loss, which, however, may favor rote memorization rather than learning patterns. The results provide some evidence for our hypothesis and suggest a possible explanation for the behaviour of Muon, that may be of interest for future explorations on how optimizers shape and affect the trained model.

Table 3: The memorization accuracy and final train loss of models trained by different optimizers. We employ different input context lengths and completion lengths to test the exact match accuracy, e.g., $(128, 8)$ means input context length $128$ and generation length $8$. All entries use the final step of the runs in Figure 12.

| Optimizer | Memorization Accuracy Clean | | | Memorization Accuracy Corrupted | | | Train Loss |
|---|---|---|---|---|---|---|---|
| | $(128, 1)$ | $(128, 8)$ | $(128, 16)$ | $(128, 1)$ | $(128, 8)$ | $(128, 16)$ | |
| Muon | 88.9% | 42.9% | 24.5% | **64.3%** | **11.1%** | **4.7%** | 1.143 |
| AdamW | 90.5% | 49.9% | 30.4% | 55.4% | 6.5% | 2.2% | 1.344 |
| SGD | **93.3%** | **60.3%** | **41.1%** | 35.2% | 1.4% | 0.4% | 2.310 |

**Remark 3** (A better finetuning approach for Muon would be helpful)**.** *For Muon pretrained models, we should apply full finetuning with Muon in order to obtain the best learning-forgetting tradeoff based on optimizer-model consistency, while Muon in SFT may potentially hurt the overall performance, which creates a dilemma. Therefore, a better finetuning approach for Muon that can give a better tradeoff for both sides would be helpful for improving the method in LLM training.*

## 5 Conclusions

In this paper, we present and analyze the optimizer-model consistency phenomenon. Based on the understanding of it, we suggest that full finetuning with the same (family of) optimizer as pretraining should be employed in SFT to achieve the best learning-forgetting tradeoff and also conduct a case study on the performance of Muon throughout the pretrain-SFT paradigm. The major limitation of the work is that the experiments are mainly conducted on small models due to the requirement of pretraining and limited computational resources, and the theory is obtained under approximations. We will be interested in scaling up our experiments and exploring more optimizers for future work.

## AI USE STATEMENT

In this work, we used generative AI tools for implementing the methods, assisting in organizing and analyzing experiment results, assisting in finding related literature and materials, and helping with grammar check and paragraph edits, and discussing about the theoretical claim. We have not used generative AI tools for developing and finding research questions and ideas. We have reviewed all AI-assisted work, checking the code and experiment results are consistent with the claims. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## REPRODUCIBILITY STATEMENT

Our experiments are run on open-source codebases LlamaFactory and NanogGPT. The experiments and methodology are presented in the main paper, and the experiment details are presented in the appendix.

## REFERENCES

Kwangjun Ahn, Noah Amsel, and John Langford. Dion2: A simple method to shrink matrix in muon. *arXiv preprint 2512.16928*, 2025a.

Kwangjun Ahn, Byron Xu, Natalie Abreu, and John Langford. Dion: Distributed orthonormalized updates. *arXiv preprint: 2504.05295*, 2025b.

Kang An, Yuxing Liu, Rui Pan, Yi Ren, Shiqian Ma, Donald Goldfarb, and Tong Zhang. Asgo: Adaptive structured gradient optimization. *arXiv preprint arXiv:2503.20762*, 2025.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International conference on machine learning*, pp. 233–242. PMLR, 2017.

Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *International Conference on Machine Learning*, pp. 404–413. PMLR, 2018.

Lukas Balles, Fabian Pedregosa, and Nicolas Le Roux. The geometry of sign gradient descent. *arXiv preprint arXiv:2002.08056*, 2020.

Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of machine learning research*, 3(Nov):463–482, 2002.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pp. 560–569. PMLR, 2018.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*, 2022.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

Ronan Collobert. Large scale machine learning. Technical report, Université de Paris VI, 2004.

Michael Crawshaw, Mingrui Liu, Francesco Orabona, Wei Zhang, and Zhenxun Zhuang. Robustness to unbounded smoothness of generalized signsgd. *Advances in neural information processing systems*, 35:9955–9968, 2022.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Aaron Defazio. Why gradients rapidly increase near the end of training. *arXiv preprint arXiv:2506.02285*, 2025.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled. *Advances in Neural Information Processing Systems*, 37: 9974–10007, 2024.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

E. Edgington and P. Onghena. *Randomization Tests*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 2007. ISBN 9781584885894.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

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

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Matthew Jagielski, Om Thakkar, Florian Tramer, Daphne Ippolito, Katherine Lee, Nicholas Carlini, Eric Wallace, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, et al. Measuring forgetting of memorized training examples. *arXiv preprint arXiv:2207.00099*, 2022.

Ruichen Jiang, Devyani Maladkar, and Aryan Mokhtari. Provable complexity improvement of adagrad over sgd: Upper and lower bounds in stochastic non-convex optimization. *arXiv preprint arXiv:2406.04592*, 2024.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Andrej Karpathy. nanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Team Kimi, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. *arXiv preprint arXiv:2304.13960*, 2023.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Yuxing Liu, Rui Pan, and Tong Zhang. Adagrad under anisotropic smoothness. *arXiv preprint arXiv:2406.15244*, 2024.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

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

Nikhil Shivakumar Nayak, Krishnateja Killamsetty, Ligong Han, Abhishek Bhandwaldar, Prateek Chanda, Kai Xu, Hao Wang, Aldo Pareja, Oleg Silkin, Mustafa Eyceoz, et al. Sculpting subspaces: Constrained full fine-tuning in llms for continual learning. *arXiv preprint arXiv:2504.07097*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Rui Pan, Yang Luo, Yuxing Liu, Yang You, and Tong Zhang. Unbiased gradient low-rank projection. *arXiv preprint arXiv:2510.17802*, 2025.

Madhur Panwar, Gail Weiss, Navin Goyal, and Antoine Bosselut. For better or for worse, transformers seek patterns for memorization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained lmos. *arXiv preprint arXiv:2502.07529*, 2025.

Fuli Qiao and Mehrdad Mahdavi. Learn more, but bother less: parameter efficient continual learning. *Advances in Neural Information Processing Systems*, 37:97476–97498, 2024.

Xingyu Qu, Peigeng Huang, and Samuel Horvath. Can Muon fine-tune Adam-pretrained models? *arXiv preprint arXiv:2605.10468*, 2026.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Yi Ren and Donald Goldfarb. Tensor normal training for deep learning models. *Advances in Neural Information Processing Systems*, 34:26040–26052, 2021.

Mark Rofin, Aditya Varre, and Nicolas Flammarion. (how) learning rates regulate catastrophic overtraining. *arXiv preprint arXiv:2604.13627*, 2026.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

Avi Schwarzschild, Zhili Feng, Pratyush Maini, Zachary Lipton, and J Zico Kolter. Rethinking llm memorization through the lens of adversarial compression. *Advances in Neural Information Processing Systems*, 37:56244–56267, 2024.

Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining. *arXiv preprint arXiv:2509.01440*, 2025.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International conference on machine learning*, pp. 4596–4604. PMLR, 2018.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. Lora vs full fine-tuning: An illusion of equivalence. *arXiv preprint arXiv:2410.21228*, 2024.

Sidak Pal Singh, Gregor Bachmann, and Thomas Hofmann. Analytic insights into structure and rank of neural network hessian maps. *Advances in Neural Information Processing Systems*, 34: 23914–23927, 2021.

Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained language models are harder to fine-tune. *arXiv preprint arXiv:2503.19206*, 2025.

Matthew Streeter and H Brendan McMahan. Less regret via online conditioning. *arXiv preprint arXiv:1002.4862*, 2010.

Weijie Su. Isotropic curvature model for understanding deep learning optimization: Is gradient orthogonalization optimal? *arXiv preprint arXiv:2511.00674*, 2025.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

Kushal Tirumala, Aram Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. *Advances in Neural Information Processing Systems*, 35:38274–38290, 2022.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vladimir Naumovich Vapnik. Adaptive and learning systems for signal processing communications, and control. *Statistical learning theory*, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam. *arXiv preprint arXiv:2409.11321*, 2024.

Shuche Wang, Fengzhuo Zhang, Jiaxiang Li, Cunxiao Du, Chao Du, Tianyu Pang, Zhuoran Yang, Mingyi Hong, and Vincent YF Tan. Muon outperforms adam in tail-end associative memory learning. *arXiv preprint arXiv:2509.26030*, 2025.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

Kaiyue Wen, Xingyu Dang, Kaifeng Lyu, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them 2.1: Hyperball optimization, 12 2025a. URL https://tinyurl.com/muonh.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. *arXiv preprint arXiv:2509.02046*, 2025b.

Martin Wistuba, Prabhu Teja Sivaprasad, Lukas Balles, and Giovanni Zappella. Continual learning with low rank adaptation. *arXiv preprint arXiv:2311.17601*, 2023.

Shuo Xie, Tianhao Wang, Sashank Reddi, Sanjiv Kumar, and Zhiyuan Li. Structured preconditioners in adaptive optimization: A unified analysis. *arXiv preprint arXiv:2503.10537*, 2025.

Tian Xie, Haoming Luo, Haoyu Tang, Yiwen Hu, Jason Klein Liu, Qingnan Ren, Yang Wang, Wayne Xin Zhao, Rui Yan, Bing Su, et al. Controlled llm training on spectral sphere. *arXiv preprint arXiv:2601.08393*, 2026.

Ruihan Xu, Jiajin Li, and Yiping Lu. On the width scaling of neural optimizers under matrix operator norms i: Row/column normalization and hyperparameter transfer. *arXiv preprint arXiv:2603.09952*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Dingzhi Yu, Rui Pan, Yuxing Liu, and Tong Zhang. Stosignsgd: Unbiased structural stochasticity fixes signsgd for training large language models. *arXiv preprint arXiv:2604.15416*, 2026.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Huizhuo Yuan, Yifeng Liu, Shuang Wu, Xun Zhou, and Quanquan Gu. Mars: Unleashing the power of variance reduction for training large models. *arXiv preprint arXiv:2411.10438*, 2024.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. Pmlr, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024a.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024b.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

## A Related Work

**Optimizers for Deep Learning.** The current workhorse of optimizers for training deep learning models, or more specifically, transformers (Vaswani et al., 2017), is undoubtedly adaptive gradient methods, including AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017) and many variants (Duchi et al., 2011; Streeter & McMahan, 2010; Tieleman, 2012; Zeiler, 2012; Shazeer & Stern, 2018; Reddi et al., 2019; You et al., 2019; Zhang et al., 2024b; Defazio et al., 2024; Yuan et al., 2024; Defazio, 2025; Bernstein et al., 2018; Crawshaw et al., 2022; Chen et al., 2023; Yu et al., 2026), to mention a few. Recently, optimizers aware of the matrix structure of the weights have emerged as a strong opponent to the adaptive gradient methods, showing great potential in training transformers. KFAC (Martens & Grosse, 2015) and Shampoo (Gupta et al., 2018) are two pioneering works utilizing the neural network structure in optimizer design, which is further explored in more recent works (Ren & Goldfarb, 2021; Zhao et al., 2024; Jordan et al., 2024; Vyas et al., 2024; Liu et al., 2025; An et al., 2025; Xie et al., 2025; Pan et al., 2025; Xie et al., 2026; Wen et al., 2025a), to mention a few, in either saving memory footprint or improving training performance. The idea has become particularly popular after Muon (Jordan et al., 2024; Liu et al., 2025) has been applied to training real frontier models (Kimi et al., 2026; Zeng et al., 2025; DeepSeek-AI, 2026). To compare and understand the performance of optimizers, research on empirically benchmarking and tracking the performance (Wen et al., 2025b; Semenov et al., 2025; Wang et al., 2025) or theoretically analyzing the convergence properties (Duchi et al., 2011; Liu et al., 2024; Jiang et al., 2024; Su, 2025) has emerged, but mainly focused on pretraining or one training setting rather than throughout the pretrain-SFT paradigm.

**Learning-Forgetting Tradeoff.** Models tend to lose the information learned from a previous task when training on a new task. This phenomenon, termed catastrophic forgetting (McCloskey & Cohen, 1989; McClelland et al., 1995; Kirkpatrick et al., 2017), is a central problem in continual learning and multitask learning. Researchers have proposed numerous methods to find a better tradeoff between learning the new task and forgetting the old task, including introducing regularization (Kirkpatrick et al., 2017; Zenke et al., 2017; Li & Hoiem, 2017) and replaying samples in previous tasks (Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017; Rolnick et al., 2019). Studies have

been conducted to examine LoRA (Hu et al., 2022) for finding a better learning-forgetting tradeoff. Biderman et al. (2024) claims that LoRA forgets less than full finetuning, although it also learns less, focusing on continual pretraining and SFT. However, on sequential continual learning settings, there is simultaneously evidence showing that LoRA can resist forgetting (Wistuba et al., 2023; Qiao & Mahdavi, 2024) and LoRA may not forget less (Shuttleworth et al., 2024) in varying settings and viewpoints. Springer et al. (2025); Rofin et al. (2026) also discuss the forgetting in SFT and draw a link to the pretraining length and SFT learning rate choice.

**Relation to the closest observations.** Liu et al. (2025) compare AdamW and Muon across pretraining and SFT and observe benefits from retaining the pretraining optimizer. Biderman et al. (2024) also study learning-forgetting tradeoffs, reporting task-dependent results, including settings favoring full finetuning on math. Their main curves vary training duration after selecting learning rates. Shuttleworth et al. (2024) show that LoRA's forgetting behavior depends on the comparison setting, while Rofin et al. (2026) study learning-rate effects at matched SFT loss. Concurrent work by Qu et al. (2026) relates optimizer mismatch to implicit bias and shows that LoRA reduces the Adam–Muon performance gap. Our question is how LoRA compares with full finetuning using the matching optimizer along the learning-forgetting frontier. We jointly consider learning rates and checkpoints across pretraining and finetuning optimizers, and relate these comparisons to activation properties and a conditional theoretical explanation.

**Memorization and Generalization.** Memorization and generalization of the model, and their relations, are one of the fundamental problems of machine learning. Traditional statistical learning theory provides viewpoints from VC dimension (Vapnik, 1998), Rademacher complexity (Bartlett & Mendelson, 2002), and uniform stability (Bousquet & Elisseeff, 2002). Zhang et al. (2016) provides evidence that deep learning models can easily memorize data while generalizing well with the randomization test methodology applied to image classification tasks, showing that such learning tasks may go beyond the scope of the traditional theory. Building upon the methodology, Arpit et al. (2017) further shows that simple patterns are learned first by the neural networks compared to noise, implying that model learning and generalization is something more than rote memorization. Recent work also extends the study to modern large foundation models (Carlini et al., 2021; 2022; Tirumala et al., 2022; Jagielski et al., 2022; Schwarzschild et al., 2024; Panwar et al., 2025), revealing an interesting but complex mechanism of the memorization and generalization of the models.

# B Experiment Settings

## B.1 Experiment Settings in Section 2

The GPT-2 pretraining experiments are done with 4 NVIDIA RTX A6000 GPUs with 48 GB of memory, GPT-2 SFT experiments with 1 A6000 GPU, and Llama-2-7B finetuning experiments with 4 A6000 GPUs. All experiments are conducted on bf16 precision.

For GPT-2 experiments, we adopt a training codebase modified from the nanoGPT codebase (Karpathy, 2022). For Llama-2-7B experiments, we employ the LlamaFactory (Zheng et al., 2024) codebase and Muon fsdp2 implementation from the Dion codebase (Ahn et al., 2025b;a).

**Training Algorithms.** In this paper, we typically consider three optimizers: AdamW, Muon, and SGD. For the SFT stage, we consider both full finetuning with the three optimizers and LoRA (Hu et al., 2022) with the optimizer matching pretraining. Specifically, we use the RMSNorm-aligned version of Muon as suggested in (Liu et al., 2025). We also apply a similar update RMSNorm alignment to SGD in this paper, i.e., the update of SGD for a layer $W \in \mathbb{R}^{m\times n}$ is $W \leftarrow W - \eta \cdot 0.2 \cdot \sqrt{mn}g/\left\|g\right\|$, where $g$ is the momentum.

**Model & Pretraining.** For the model, we adopt the same architecture and parameter number as the GPT-2-small model (Ouyang et al., 2022) (approximately 123M), changing only the activation function from GELU (Hendrycks & Gimpel, 2016) to SwiGLU (Shazeer, 2020). We train the models from scratch with 8B tokens on the OpenWebText dataset (Gokaslan & Cohen, 2019). For pretraining, we adopt AdamW and Muon with learning rate 3e-3.

**SFT.** We consider three different types of tasks for SFT: MetaMathQA (Yu et al., 2023) for math, Alpaca (Taori et al., 2023) for instruction following, Magicoder-Evol-Instruct-110K (Wei et al., 2023) for coding. We employ full finetuning and LoRA with $r = 4, 8, 32, 128$ and $\alpha = 2r$ following

the suggesion in Biderman et al. (2024). For each training algorithm, we apply a grid search for learning rates, with other hyperparameters fixed. The grid always contains the learning rate that achieves the best validation loss in the SFT task, and is listed in Appendix B.1.1.

**Evaluation.** We care about two major metrics for evaluation: the OpenWebText validation loss as a measure for forgetting, i.e., how much the models preserve the pretraining knowledge, and the loss on the new task validation loss as a measure for learning, i.e., how much the models learn the new ability (lower is better). For a comprehensive and fair comparison between the training algorithms, we plot the Pareto frontier of the learning-forgetting tradeoff, with $x$-axis denoting the forgetting metric and $y$-axis denoting the learning metric. For each training algorithm, we consider all the runs in the learning rate grid search, and for a single run, we evaluate intermediate checkpoints. Then, we gather all the data points for each training algorithm to present the Pareto frontier to find the limit of this training algorithm in learning the most and forgetting the least. When plotting the Pareto frontier figures, we use a robust frontier selection strategy for clarity of the figures. For figures with loss as measurements, for each point $(x_i, y_i)$ in the Pareto frontier of a specific algorithm, we plot it as the frontier only when $\{x_i \leq x_j - c$ or $y_i \leq y_j - c\}$ holds for all other $(x_j, y_j)$ in this specific algorithm, where $c = 0.001$, meaning that we only include the points with a significant improvement compared to others. We apply this plotting strategy to figures with accuracy as measurements similarly with $c = 0.001$.

**Llama-2-7B Settings.** We also verify the results in finetuning Llama-2-7B with MetaMathQA, following the MATH IFT scheme in Biderman et al. (2024). We follow the same logic as the GPT-2 experiment to do learning rate grid search and plot the Pareto frontier of different training algorithms, changing only the measures of forgetting and learning from loss to (benchmark-averaged) accuracy (higher is better).

### B.1.1 Detailed Experiment Settings

**GPT-2 Pretraining.** GPT-2 models pretrained on OpenWebText dataset.

- `sequence length`: 1024
- `effective batch size`: 240 (4 GPUs $\times$ 5 gradient accumulation $\times$ 12 batch size)
- `token`: 8B
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `weight decay`: 0.1
- `learning rate`: 3e-3 for both AdamW and Muon

**GPT-2 Alpaca.** GPT-2 models (pretrained by AdamW and Muon) finetuned on Alpaca.

- `sequence length`: 1024
- `effective batch size`: 128 (1 GPU $\times$ 4 gradient accumulation $\times$ 32 batch size)
- `iterations`: 800 (approximately 2 epochs)
- `evaluation interval`: every 200 iterations
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `full finetuning weight decay`: 0
- `full finetuning dropout`: 0.1
- `full finetuning learning rate grid`: {3e-5, 5e-5, 1e-4, 3e-4, 5e-4} for both AdamW and Muon

**GPT-2 MetaMathQA.** GPT-2 models (pretrained by AdamW and Muon) finetuned on MetaMathQA.

- `sequence length`: 1024

- `effective batch size`: 128 (1 GPU $\times$ 4 gradient accumulation $\times$ 32 batch size)
- `iterations`: 6100 (approximately 2 epochs)
- `evaluation interval`: every 300 iterations
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `full finetuning weight decay`: 0
- `full finetuning dropout`: 0.1
- `full finetuning learning rate grid`: {5e-5, 1e-4, 3e-4, 5e-4} for both AdamW and Muon
- `LoRA dropout`: 0.1
- `LoRA alpha`: $2r$ for each $r = 4, 8, 32, 128$
- `LoRA learning rate grid`: {5e-4,1e-3,3e-3,5e-3,1e-2} for both AdamW and Muon

**GPT-2 Magicoder.** GPT-2 models (pretrained by AdamW and Muon) finetuned on Magicoder.

- `sequence length`: 1024
- `effective batch size`: 128 (1 GPU $\times$ 4 gradient accumulation $\times$ 32 batch size)
- `iterations`: 1800 (approximately 2 epochs)
- `evaluation interval`: every 200 iterations
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `full finetuning weight decay`: 0
- `full finetuning dropout`: 0.1
- `full finetuning learning rate grid`: {5e-5, 1e-4, 3e-4, 5e-4, 1e-3} for both AdamW and Muon

**Llama-2 MetaMathQA.** Llama-2-7B-hf[1] finetuned on MetaMathQA.

- `sequence length`: 1024
- `effective batch size`: 128 (4 GPU $\times$ 8 gradient accumulation $\times$ 4 batch size)
- `iterations`: 3055 (approximately 1 epoch)
- `evaluation interval`: every 500 iterations
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `full finetuning weight decay`: 0
- `full finetuning dropout`: 0
- `full finetuning learning rate grid`: {1e-5,2e-5,5e-5,1e-4} for both AdamW and Muon
- `LoRA dropout`: 0
- `LoRA alpha`: $2r$ for each $r = 16, 64, 256$
- `LoRA learning rate grid`: {5e-5,1e-4,2e-4,5e-4}

**RNJ-1 MetaMathQA.** RNJ-1[2] (8.3B, pretrained by Muon) finetuned on MetaMathQA.

- `sequence length`: 1024, with sequence packing

[1] https://huggingface.co/meta-llama/Llama-2-7b-hf
[2] https://huggingface.co/EssentialAI/rnj-1

- `effective batch size`: 64 (4 GPUs $\times$ 1 gradient accumulation $\times$ 16 batch size)
- `iterations`: 1398 (1 epoch)
- `evaluation interval`: every 250 iterations and at the final checkpoint
- `scheduler`: cosine (Loshchilov & Hutter, 2016), with 10% warmup
- `gradient clipping`: norm (threshold=1)
- `weight decay`: 0 for both full finetuning and LoRA
- `full finetuning learning rate grid`: {2e-6,5e-6,1e-5,2e-5} for both AdamW and Muon
- `LoRA optimizer`: Muon
- `LoRA dropout`: 0
- `LoRA alpha`: $2r$ for each $r = 16, 64, 256$
- `LoRA learning rate grid`: {5e-5,1e-4,2e-4,5e-4}

**Additional hyperparameter settings of optimizers.** The following hyperparameters are fixed in the experiments.

- AdamW:
  - $(\beta_1, \beta_2)$: (0.9,0.95)
  - $\epsilon$: 1e-8
- Muon:
  - momentum: 0.9
  - AdamW $(\beta_1, \beta_2)$: (0.9,0.95)
  - AdamW $\epsilon$: 1e-8
- SGD and other optimizers (Algorithm 1,2):
  - momentum: 0.9

### B.2 Experiment Settings in Section 4.1

**Dataset & Training.** We use corrupted datasets constructed from the OpenWebText dataset (Gokaslan & Cohen, 2019), tokenized by the standard GPT-2 tokenizer. To obtain the target dataset, we first retrieve a subset of 40M tokens from the original dataset. Then we split the tokens into blocks of length 1024. For each block, we do a random permutation, i.e., randomly reorder the tokens, with probability $\alpha = 0.5$. We call a block corrupted if it is randomly permuted. In this way, we keep the vocabulary frequency the same, but destroy the patterns in the corrupted blocks, ensuring that models can complete these blocks only by memorization. We use the same model configuration and optimizers as Section 2 for GPT-2, and set the sequence length to 512, with an effective batch size of 256. Following a learning rate search, we report Muon and AdamW with learning rate $10^{-3}$ and SGD with $5 \times 10^{-4}$. All three runs use microbatch size 64, four gradient accumulation steps, and 30,000 training iterations. Table 3 and Figure 12 use the same run for each optimizer, with all table entries taken at iteration 30,000.

**Evaluation.** To evaluate the memorization accuracy of models, we test the exact match accuracy of the model generation compared to the original text. Specifically, we randomly sample a subset from the original training dataset to form the evaluation set. By fixing an input length and generation length pair $(a, b)$, we test whether the model can generate exactly the same sequence as the original context of length $b$ with the input of length $a$ using greedy decoding and compute the accuracy.

### B.3 Additional Optimizers Implementation

In Figure 5, we involve two additional optimizers $\mathcal{A}_{1,1}$ and $\mathcal{A}_{\infty,\infty}$, which are derived from the framework (1) with norm $\|\cdot\|_{1,1}$ and $\|\cdot\|_{\infty,\infty}$. For an arbitrary matrix $Z \in \mathbb{R}^{m\times n}$, it holds that

$$\|Z\|_{1,1} = \max_{i\in[m]} \sum_{j=1}^{n} |Z_{i,j}| \quad \text{and} \quad \|Z\|_{\infty,\infty} = \max_{j\in[n]} \sum_{i=1}^{m} |Z_{i,j}| .$$

We then solve (1) for $\|\cdot\|_{1,1}$. Notice that

$$\langle Z, G\rangle = \sum_{i=1}^{m}\sum_{j=1}^{n} Z_{i,j}G_{i,j} \le \sum_{i=1}^{m}\sum_{j=1}^{n} |Z_{i,j}| \cdot \max_{j\in[n]} |G_{i,j}| \le \left(\max_{i\in[m]} \sum_{j=1}^{n} |Z_{i,j}|\right)\left(\sum_{i=1}^{m}\max_{j\in[n]} |G_{i,j}|\right)$$
$$= \|Z\|_{1,1} \cdot \left(\sum_{i=1}^{m}\max_{j\in[n]} |G_{i,j}|\right),$$

where $Z \triangleq W_{t+1} - W_t$. We note that the inequality holds as an equality when for each $i \in [m], j \in [n]$

$$\mathcal{A}_{1,1} \text{ Update:} \quad Z_{i,j} = \begin{cases} \frac{R}{k_i}, & \text{if } |G_{i,j}| = \max_{l\in[n]} |G_{i,l}|, \\ 0, & \text{else,} \end{cases}$$

where $k_i$ denotes the number of entries $G_{i,j}$ such that $|G_{i,j}| = \max_{l\in[n]} |G_{i,l}|$ for a fixed $i \in [m]$ and $R$ is the update scale. Similarly, we can also derive the algorithm under $\|\cdot\|_{\infty,\infty}$.

$$\mathcal{A}_{\infty,\infty} \text{ Update:} \quad Z_{i,j} = \begin{cases} \frac{R}{k_j}, & \text{if } |G_{i,j}| = \max_{l\in[m]} |G_{l,j}|, \\ 0, & \text{else,} \end{cases}$$

where $k_j$ denotes the number of entries $G_{i,j}$ such that $|G_{i,j}| = \max_{l\in[m]} |G_{l,j}|$ for a fixed $j \in [n]$ and $R$ is the update scale. Intuitively, the update formula means that we only update the entries with the largest gradient scale in every row for $\mathcal{A}_{1,1}$ and every column for $\mathcal{A}_{\infty,\infty}$ in each iteration. Then, by incorporating momentum and the update RMS norm alignment (Liu et al., 2025), we formally write the two algorithms in Algorithm 1 and 2. We also include our version of SGD in Algorithm 3.

**Algorithm 1** Row Max Descent ($\mathcal{A}_{1,1}$)

1: **Input:** $W_0 \in \mathbb{R}^{m\times n}$, the number of iterations $T$, schedule $\{\eta_t\}_{t=0}^{T-1}$, momentum $\beta$
2: Initialize $M_{-1} = 0 \in \mathbb{R}^{m\times n}$
3: **for** $t = 0$ **to** $T-1$ **do**
4: Obtain gradient $G_t$ at $W_t$
5: $M_t = \beta M_{t-1} + (1-\beta)G_t$
6: Compute $U_t$ by $[U_t]_{i,j} = \begin{cases} \frac{R}{k_i}, & \text{if } |[M_t]_{i,j}| = \max_{l\in[n]} |[M_t]_{i,l}|, \\ 0, & \text{else.} \end{cases}$, where $k_i$ denotes the number of $M_t$ entries such that $|[M_t]_{i,j}| = \max_{l\in[m]} |[M_t]_{i,l}|$ for a fixed $j$
7: $W_{t+1} = W_t - \eta_t \cdot 0.2 \cdot \frac{U_t}{\|U_t\|_{\text{RMS}}}$
8: **end for**

**Algorithm 2** Column Max Descent ($\mathcal{A}_{\infty,\infty}$)

1: **Input:** $W_0 \in \mathbb{R}^{m\times n}$, the number of iterations $T$, schedule $\{\eta_t\}_{t=0}^{T-1}$, momentum $\beta$
2: Initialize $M_{-1} = 0 \in \mathbb{R}^{m\times n}$
3: **for** $t = 0$ **to** $T-1$ **do**
4: Obtain gradient $G_t$ at $W_t$
5: $M_t = \beta M_{t-1} + (1-\beta)G_t$
6: Compute $U_t$ by $[U_t]_{i,j} = \begin{cases} \frac{R}{k_j}, & \text{if } |[M_t]_{i,j}| = \max_{l\in[m]} |[M_t]_{l,j}|, \\ 0, & \text{else.} \end{cases}$, where $k_j$ denotes the number of $M_t$ entries such that $|[M_t]_{i,j}| = \max_{l\in[m]} |[M_t]_{l,j}|$ for a fixed $j$
7: $W_{t+1} = W_t - \eta_t \cdot 0.2 \cdot \frac{U_t}{\|U_t\|_{\text{RMS}}}$
8: **end for**

**Algorithm 3** SGD (with update RMS norm alignment)

**Input:** $W_0 \in \mathbb{R}^{m\times n}$, the number of iterations $T$, schedule $\{\eta_t\}_{t=0}^{T-1}$, momentum $\beta$
Initialize $M_{-1} = 0 \in \mathbb{R}^{m\times n}$
**for** $t = 0$ **to** $T-1$ **do**
  Obtain gradient $G_t$ at $W_t$
  $M_t = \beta M_{t-1} + (1-\beta)G_t$
  $U_t = M_t$
  $W_{t+1} = W_t - \eta_t \cdot 0.2 \cdot \frac{U_t}{\|U_t\|_{\mathrm{RMS}}}$
**end for**

# C Additional Experiment Results

## C.1 Evaluation Settings

For both models in Figure 1, task capability is GSM8K exact-match accuracy using the strict-match filter. General capability is the mean of HellaSwag and ARC-Challenge normalized accuracy and WinoGrande accuracy. Frontiers are constructed across evaluated learning rates and checkpoints.

## C.2 SignSGD on GPT-2

We add SignSGD with momentum 0.9, using the same RMS update scale as SGD and no Nesterov momentum. Figure 7 compares full finetuning on MetaMathQA from both pretrained GPT-2 checkpoints, with a common learning rate grid $\{10^{-4}, 3\times10^{-4}, 5\times10^{-4}\}$, seed 1337, and evaluations every 300 updates through step 6000. Other settings follow Appendix B.1.1. SignSGD achieves a tradeoff comparable to AdamW on the AdamW-pretrained model, consistent with their shared $\mathcal{A}_{1,\infty}$ family. On the Muon-pretrained model, Muon remains favorable over much of the frontier. These observations support the optimizer-family prediction, without testing the theorem's idealized assumptions directly.




Figure 7: SignSGD compared with AdamW, Muon, and SGD on GPT-2/MetaMathQA. Left: AdamW-pretrained checkpoint; right: Muon-pretrained checkpoint. Lower is better on both axes. All methods use the same three learning rates and evaluation steps.

## C.3 Variation across SFT Seeds

We repeat GPT-2/MetaMathQA SFT with two additional random seeds (1338 and 1339), keeping each pretrained checkpoint fixed. Table 4 reports mean ± sample standard deviation across these two runs ($n = 2$), for full finetuning with learning rate $10^{-4}$ and LoRA with rank 32, $\alpha = 64$, and learning rate $10^{-3}$. Both methods use the matching optimizer and effective batch size 128; full FT uses microbatch size 32 with four accumulation steps, and LoRA uses 16 with eight. Variation is small relative to the Full FT–LoRA differences at these reported steps. These statistics describe seed variability at fixed settings, rather than confidence intervals for an entire Pareto frontier.

Table 4: Additional SFT seeds on MetaMathQA. Each entry is (OWT loss, task loss), with mean $\pm$ sample standard deviation ($n = 2$). The base column gives the optimizer used for both pretraining and SFT.

| Base | Step | Full FT | LoRA ($r = 32$) |
|---|---|---|---|
| AdamW | 600 | $(3.1495 \pm 0.0031, 0.7226 \pm 0.0060)$ | $(3.2514 \pm 0.0093, 0.7713 \pm 0.0119)$ |
| AdamW | 3000 | $(3.4705 \pm 0.0022, 0.4373 \pm 0.0014)$ | $(3.7941 \pm 0.0022, 0.4778 \pm 0.0019)$ |
| AdamW | 6000 | $(3.5375 \pm 0.0006, 0.3847 \pm 0.0014)$ | $(3.9187 \pm 0.0123, 0.4196 \pm 0.0038)$ |
| Muon | 600 | $(3.2545 \pm 0.0085, 0.6964 \pm 0.0077)$ | $(3.3956 \pm 0.0052, 0.7392 \pm 0.0098)$ |
| Muon | 3000 | $(3.6793 \pm 0.0105, 0.4007 \pm 0.0009)$ | $(4.0945 \pm 0.0015, 0.4760 \pm 0.0005)$ |
| Muon | 6000 | $(3.7837 \pm 0.0091, 0.3534 \pm 0.0010)$ | $(4.2271 \pm 0.0074, 0.4147 \pm 0.0037)$ |

### C.4 More Results for LoRA Comparing to Full Finetuning

In this section, we provide some additional experiment results specifically on LoRA under the same settings as Section 2.

**Reproduction of the Observations in Biderman et al. (2024).** We can reproduce the observations in Biderman et al. (2024) with our Llama results, as presented in Figure 8, where we can observe in the left figure that LoRA learns less (lower math score) and forgets less (higher general score) if we all choose the learning rates leading to the best math performance. Here, LoRA rank 64 shows a greater forgetting compared to rank 256, mainly because rank 256 diverges for lr=5e-4, and a smaller learning rate leads to less forgetting. In the right figure of Figure 8, we choose a smaller learning rate instead, and can find that the conclusions are different. With a smaller learning rate, full finetuning tends to learn more and forget less compared to LoRA.

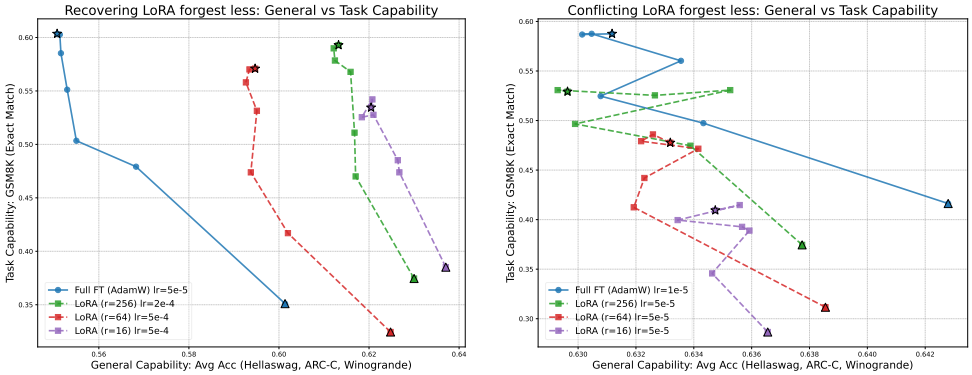



Figure 8: Reproduction of the results in Biderman et al. (2024). In the left figure, we choose only the learning rate leading to the best math performance for each training algorithm instead of presenting a Pareto frontier. In the right figure, we choose some smaller learning rates for comparison.

**Learning Rate is Important in Learning-Forgetting Tradeoff.** In Figure 9, we show how learning rate affects the learning-forgetting tradeoff. Generally, a larger learning rate achieves a relatively higher learning score while leading to much more significant forgetting, which also aligns with the observations in Rofin et al. (2026), a concurrent work to us. Therefore, it is important to choose an appropriate learning rate in SFT based on the requirements of the final model.

### C.5 More Detailed Activation Plots

In Figures 10 and 11, we present the average activation sparsity of detailed modules and specific activation splits. We can observe that the detailed sparsity plots also generally follow our observations that optimizers regularize the activations based on the corresponding matrix-induced norm $\alpha, \beta$. Interestingly, in terms of sparsity for $\mathcal{A}_{1,\infty}$ (including AdamW and SignSGD), the input activations of the next layer can be quite different from the output activations of the last layer, which should connect. This can be related to the effect of the activation function (inside FFN), attention logits

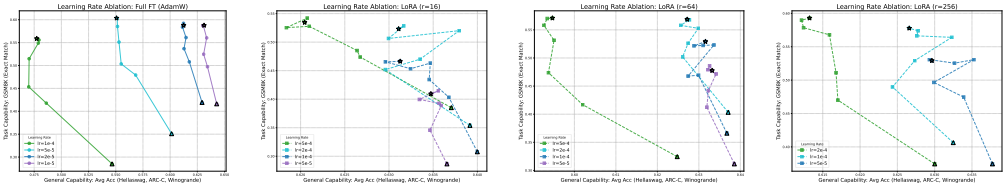



Figure 9: Learning rate affects forgetting. Note that learning rate lr=5e-4 failed for LoRA with $r = 256$.

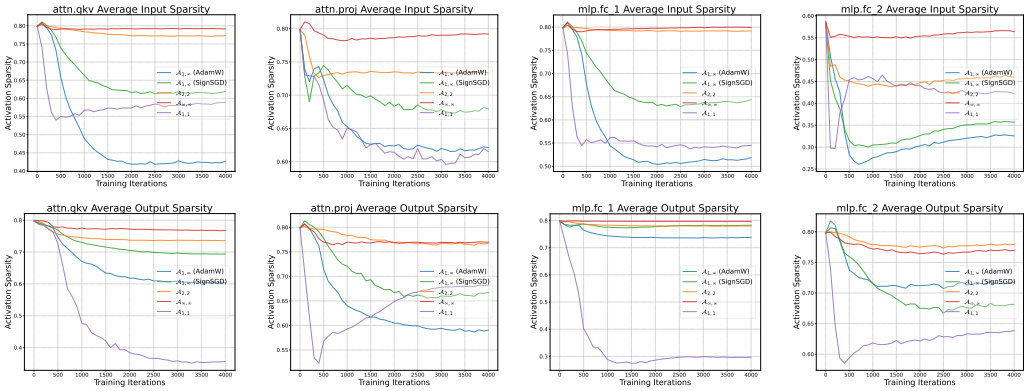



Figure 10: The average activation sparsity of each linear layer under the same settings of Figure 5. We consider 2 linear layers for attention modules: the QKV projection (attn.qkv) and attention output projection (attn.proj); 2 linear layers for FFN modules: the up projection (mlp.fc_1) and down projection (mlp.fc_2).

computation (inside attention modules), and layernorms and residual connections (between modules and layers). It would be an interesting future work to further explore the mechanism behind this phenomenon and how the activations change during training.

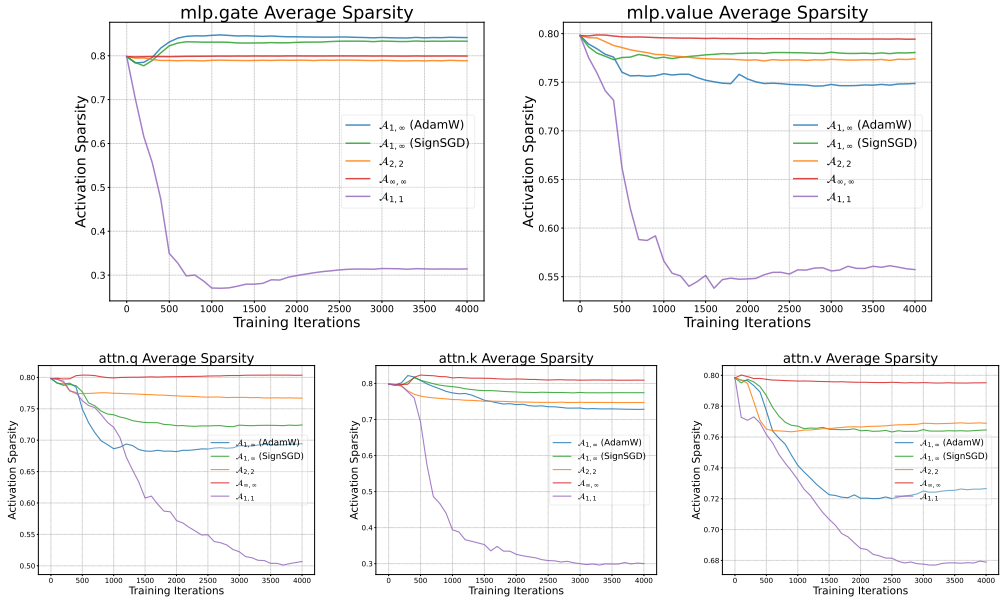



Figure 11: The average activation sparsity of some split activations under the same settings of Figure 5 and 10, including q,k,v in attention modules and gate,value in FFN modules introduced by the SwiGLU activation function.

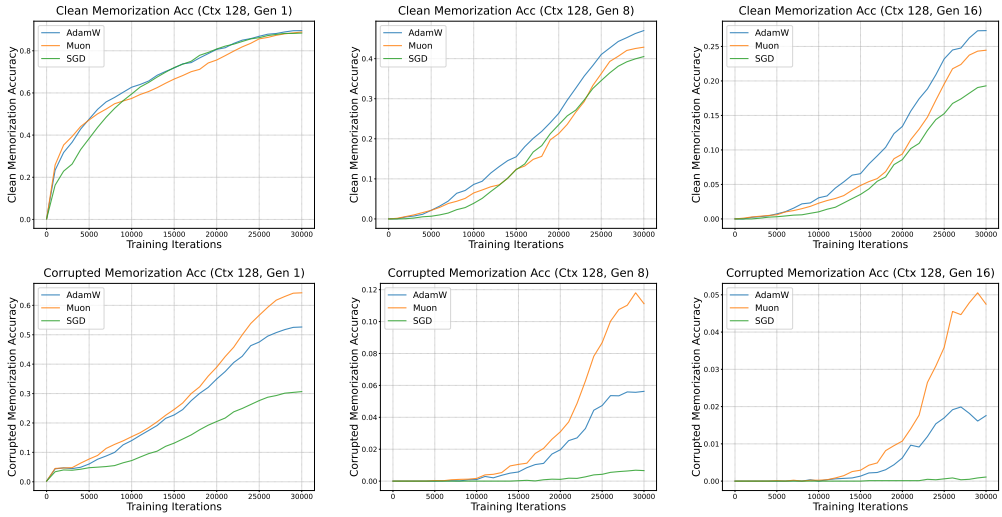



Figure 12: The memorization accuracy of clean and corrupted data under different generation lengths. Curves and Table 3 are generated from the same fixed runs.

### C.6 Plots for the Memorization Accuracy

The memorization accuracy curve with respect to iteration number is presented in Figure 12, under the same settings as Table 3.

## D Proof of Section 3

### D.1 Comparison between Norms

We begin with some auxiliary lemmas for comparing different norms. Lemma 1 is a standard result for vector norms, and Lemma 2 is a simple extension of it.

**Lemma 1.** *For an arbitrary vector $z \in \mathbb{R}^d$ and $\alpha_1, \alpha_2 \in [1, \infty]$ such that $\alpha_1 \le \alpha_2$, it holds that*

$$\|z\|_{\alpha_2} \le \|z\|_{\alpha_1} \le d^{\frac{1}{\alpha_1}-\frac{1}{\alpha_2}} \|z\|_{\alpha_2}.$$

**Lemma 2.** *For an arbitrary matrix $A \in \mathbb{R}^{m\times n}$ and $\alpha_1, \alpha_2 \in [1, \infty]$ such that $\alpha_1 \le \alpha_2$ and $\beta_1, \beta_2 \in [1, \infty]$ such that $\beta_1 \le \beta_2$, it holds that*

$$\|A\|_{\alpha_1,\beta} \le \|A\|_{\alpha_2,\beta} \le n^{\frac{1}{\alpha_1}-\frac{1}{\alpha_2}} \|A\|_{\alpha_1,\beta} \quad \text{and} \quad \|A\|_{\alpha,\beta_2} \le \|A\|_{\alpha,\beta_1} \le m^{\frac{1}{\beta_1}-\frac{1}{\beta_2}} \|A\|_{\alpha,\beta_2}$$

*Proof.* Let us denote

$$z_1 \in \operatorname*{argmax}_{z\neq 0} \frac{\|Az\|_\beta}{\|z\|_{\alpha_1}} \quad \text{and} \quad z_1 \in \operatorname*{argmax}_{z\neq 0} \frac{\|Az\|_\beta}{\|z\|_{\alpha_2}},$$

which implies that

$$\|Az_1\|_\beta = \|A\|_{\alpha_1,\beta} \|z_1\|_{\alpha_1} \quad \text{and} \quad \|Az_2\|_\beta = \|A\|_{\alpha_2,\beta} \|z_2\|_{\alpha_2}.$$

Combining with Lemma 1, we therefore have

$$\|A\|_{\alpha_1,\beta} = \frac{\|Az_1\|_\beta}{\|z_1\|_{\alpha_1}} \le \frac{\|Az_1\|_\beta}{\|z_1\|_{\alpha_2}} \le \frac{\|Az_2\|_\beta}{\|z_2\|_{\alpha_2}} = \|A\|_{\alpha_2,\beta}$$

and

$$\|A\|_{\alpha_2,\beta} = \frac{\|Az_2\|_\beta}{\|z_2\|_{\alpha_2}} \le n^{\frac{1}{\alpha_1}-\frac{1}{\alpha_2}} \frac{\|Az_2\|_\beta}{\|z_2\|_{\alpha_1}} \le n^{\frac{1}{\alpha_1}-\frac{1}{\alpha_2}} \frac{\|Az_1\|_\beta}{\|z_1\|_{\alpha_1}} = n^{\frac{1}{\alpha_1}-\frac{1}{\alpha_2}} \|A\|_{\alpha_1,\beta},$$

which concludes the proof for the first inequality. We can prove the second inequality similarly. □

### D.2 Proof of Theorem 1

*Proof.* Consider an arbitrary SFT optimizer choice $\mathcal{A}_{\alpha_2,\beta_2}$ with $\alpha_2, \beta_2 \in [1, \infty)$ and $\Delta W$ is the corresponding SFT update. From Assumption 2, for a fixed $\beta_* \in [1, \infty)$, there exists $\alpha_* \in [1, \infty)$ such that

$$\mathbb{E}\left[\|\Delta W x\|_{\beta_*}^2\right] = \Theta\left(\|\Delta W\|_{\alpha_*,\beta_*}^2 \mathbb{E}\left[\|x\|_{\alpha_*}^2\right]\right).$$

Since for all $\alpha, \beta \in [1, \infty]$, it always holds that

$$\|\Delta W x\|_\beta \le \|\Delta W\|_{\alpha,\beta} \|x\|_\alpha$$

based on the definition of matrix induced norms, $\alpha_*, \beta_*$ should correspond to the tightest upper bound to yield Assumption 2. We then discuss the best value of $\alpha$, i.e., $\alpha_*$, in this case.

- If $\alpha \in (\alpha_1, \infty]$, based on Lemma 2 and Assumption 4, it holds that

  $$\|\Delta W\|_{\alpha_1,\beta_*} \le \|\Delta W\|_{\alpha,\beta_*} \quad \text{and} \quad \mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \Theta\left(\mathbb{E}\left[\|x\|_\alpha^2\right]\right).$$

  Thus, we have

  $$\|\Delta W\|_{\alpha_1,\beta_*}^2 \mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \mathcal{O}\left(\|\Delta W\|_{\alpha,\beta_*}^2 \mathbb{E}\left[\|x\|_\alpha^2\right]\right)$$

- If $\alpha \in [1, \alpha_1)$, based on Lemma 2 and Assumption 4, it holds that

  $$\|\Delta W\|_{\alpha_1,\beta_*} \le n^{\frac{1}{\alpha}-\frac{1}{\alpha_1}} \|\Delta W\|_{\alpha,\beta_*} \quad \text{and} \quad \mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \Theta\left(n^{\frac{2}{\alpha_1}-\frac{2}{\alpha}} \mathbb{E}\left[\|x\|_\alpha^2\right]\right).$$

  Thus, we have

  $$\|\Delta W\|_{\alpha_1,\beta_*}^2 \mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \mathcal{O}\left(\|\Delta W\|_{\alpha,\beta_*}^2 \mathbb{E}\left[\|x\|_\alpha^2\right]\right)$$

From the two cases, we conclude that we should have $\alpha_* = \alpha_1$ to yield the tightest upper bound since it holds that $\|\Delta W\|_{\alpha_1,\beta_*}^2 \mathbb{E}\left[\|x\|_{\alpha_1}^2\right] = \mathcal{O}\left(\|\Delta W\|_{\alpha,\beta_*}^2 \mathbb{E}\left[\|x\|_\alpha^2\right]\right)$ for all $\alpha \in [1, \infty]$.

Moreover, based on Assumption 4 on $\Delta W x$, we have

$$\begin{aligned}\mathbb{E}\left[\|\Delta W x\|_2^2\right] &= \Theta\left(m^{-\max\{\frac{2}{\beta_2}-1,0\}} \mathbb{E}\left[\|\Delta W x\|_{\beta_2}^2\right]\right) \\ &= \Theta\left(m^{-\max\{\frac{2}{\beta_2}-1,0\}+\max\{\frac{2}{\beta_2}-\frac{2}{\beta_*},0\}} \mathbb{E}\left[\|\Delta W x\|_{\beta_*}^2\right]\right).\end{aligned}$$

This implies that by choosing $\beta_* = \beta_2$ and substituting $\alpha_* = \alpha_1$, we have

$$\mathcal{L}_{\text{forget}}(\Delta W) \triangleq \mathbb{E}\left[\|\Delta W x\|_2^2\right] = \Theta\left(m^{\max\{\frac{1}{\beta_2}-\frac{1}{2},0\}} \|\Delta W\|_{\alpha_1,\beta_2}^2 \mathbb{E}\left[\|x\|_{\alpha_1}^2\right]\right) \tag{3}$$

for $\Delta W$ derived with any choice of $\alpha_2, \beta_2 \in [1, \infty]$.

Then we consider the SFT update $\Delta W$. Based on (1) and Assumption 3, the $\Delta W$ derived by SFT optimizer $\mathcal{A}_{\alpha_2,\beta_2}$ holds

$$\Delta W(\mathcal{A}_{\alpha_2,\beta_2}) = \operatorname*{argmin}_{\|Z\|_{\alpha_2,\beta_2} \le \eta} \langle Z, G \rangle,$$

where $\eta$ denotes the update scale. Taking $C \triangleq H_0 + \min_{\|Z\|_{\alpha_2,\beta_2} \le \eta} \langle Z, G \rangle$, this is equivalent to

$$\Delta W(\mathcal{A}_{\alpha_2,\beta_2}) = \operatorname*{argmin}_{\langle Z,G \rangle \le C - H_0} \|Z\|_{\alpha_2,\beta_2} = \operatorname*{argmin}_{\mathcal{L}_{\text{SFT}}(Z) \le C} \|Z\|_{\alpha_2,\beta_2}, \tag{4}$$

which can be straightforwardly verified by the KKT condition.

With (3) and (4) in hand, we are ready to prove the theorem. For brevity, we define the following function as the bound derived in (3) for any $\alpha, \beta \in [1, \infty]$:

$$g(\alpha, \beta) \triangleq m^{\max\{\frac{1}{\beta}-\frac{1}{2},0\}} \|\Delta W(\mathcal{A}_{\alpha,\beta})\|_{\alpha_1,\beta}^2 \mathbb{E}\left[\|x\|_{\alpha_1}^2\right],$$

where $\Delta W(\mathcal{A}_{\alpha,\beta})$ denotes the SFT update if $\mathcal{A}_{\alpha,\beta}$ is employed as the SFT optimizer. Thus, it holds that $\mathcal{L}_{\text{forget}}(\Delta W(\mathcal{A}_{\alpha,\beta})) = \Theta\left(g(\alpha,\beta)\right)$. Then we have that for any $\alpha, \beta \in [1,\infty]$ pair yielding $\beta < 2$, it holds that $g(\alpha_1,\beta_2) = \mathcal{O}\left(g(\alpha,\beta)\right)$ with $\beta_2 \geq 2$, since we have

$$\|\Delta W(\mathcal{A}_{\alpha,\beta})\|_{\alpha_1,\beta} \geq \|\Delta W(\mathcal{A}_{\alpha,\beta})\|_{\alpha_1,\beta_2} \geq \min_{\langle \Delta W, G\rangle \leq C - H_0} \|\Delta W\|_{\alpha_1,\beta_2} = \|\Delta W(\mathcal{A}_{\alpha_1,\beta_2})\|_{\alpha_1,\beta_2}$$

based on Lemma 2 and (4).

When taking $\alpha_2 = \alpha_1$ and $\beta_2 \geq 2$, we simultaneously have

$$g(\alpha_1,\beta_2) = \mathcal{O}\left(g(\alpha,\beta)\right) \quad \text{and} \quad \mathcal{L}_{\text{forget}}(\Delta W(\mathcal{A}_{\alpha,\beta})) = \Theta\left(g(\alpha,\beta)\right)$$

for all $\alpha, \beta$ pair such that $\alpha \in [1,\infty]$ and $\beta \in [1,2]$. Therefore, it holds that

$$\mathcal{L}_{\text{forget}}(\Delta W(\mathcal{A}_{\alpha_1,\beta_2})) = \mathcal{O}\left(\inf_{\alpha\in[1,\infty],\beta\in[1,2]} \mathcal{L}_{\text{forget}}(\Delta W(\mathcal{A}_{\alpha,\beta}))\right),$$

which concludes the proof.

□

## E LICENSES

The Alpaca dataset (Taori et al., 2023) (https://huggingface.co/datasets/tatsu-lab/alpaca) is released under the cc-by-nc-4.0 license. The MetaMathQA (Yu et al., 2023) (https://huggingface.co/datasets/meta-math/MetaMathQA) and Magicoder (Wei et al., 2023) (https://huggingface.co/datasets/ise-uiuc/Magicoder-OSS-Instruct-75K) datasets and the NanoGPT (Karpathy, 2022) (https://github.com/karpathy/nanogpt) and Dion (Ahn et al., 2025b;a) (https://github.com/microsoft/dion) codebases are under the MIT license. The LlamaFactory codebase (Zheng et al., 2024) (https://github.com/hiyouga/LlamaFactory) is under the Apache-2.0 license. The Llama-2-7b-hf model (Touvron et al., 2023) (https://huggingface.co/meta-llama/Llama-2-7b-hf) is under the llama2 license.