Towards understanding of orthogonalization in Muon

Anonymous Authors[1]

Abstract

Muon is a recent optimizer that relies on matrix orthogonalization of updates and has been shown to improve large language model (LLM) training. It does so by introducing additional momentum and Newton-Schulz iteration to the stochastic spectral descent method (SSD). However, it incurs higher communication cost if tensor parallelism is enabled, and its hyperparameter transfer properties are not yet fully explored. We first introduce *block-wise orthogonalization*, splitting weight matrices into independent tiles that are orthogonalized separately and recombined and empirically analyze its influence on training. This retains the validation loss while allowing up to 16x tensor parallel splits of weight matrices. Second, we show that under spectral regularization a single learning rate transfers when depth, width of the model, *and* token count are co-scaled under Chinchilla guidelines. Finally, we show that a higher weight decay value of 0.1 underperforms during the first 80% of the training but outperforms lower values after that, which can be attributed to the tighter spectral norm constraint. Based on this, we propose weight decay clipping and scheduling to capture both regimes. Overall, we demonstrate experimentally for nanoGPT models from 124M to 1.4B parameters that spectral regularization, both with block-wise and full-matrix orthogonalization, allows for learning rate transfer across multiple scaling dimensions and better generalization with weight decay due to the tighter spectral norm constraint. The code is available at `https://anonymous.4open.science/r/MuonSBW-23A2`.

1. Introduction

Optimization is one of the driving forces behind the rapid development of deep learning – and LLMs in particular. It is closely connected to the scaling laws through feature learning: with specific parameterization and learning-rate scaling (maximal update parameterization, μP) (Yang et al., 2021; 2023), training remains stable, and hyperparameters, such as learning rate, transfer when scaling model size (e.g., width (Yang & Hu, 2021) or depth (Yang et al., 2024)).

Adam (Kingma & Ba, 2015) – augmented with decoupled weight decay (AdamW) (Loshchilov & Hutter, 2019) – is the default LLM optimizer, yet it has some disadvantages. First, it is heuristically derived with the need of bias correction and storing running statistics for every parameter, which as the consequence increases its complexity and memory footprint. More importantly, it is known to have instabilities during the training, known as "loss spikes" (Molybog et al., 2023), which is especially problematic when training larger models and requires regular re-starts during the training from the latest checkpoints (Chowdhery et al., 2022). Hyperparameters also fail to transfer across width unless one uses μP scaling (Littwin & Yang, 2023).

Recently, an alternative, Muon, was proposed in (Jordan et al., 2024; Bernstein & Newhouse, 2024). It enjoys a faster convergence compared to AdamW during NanoGPT speed-runs (Jordan et al., 2024) and is designed for more stable training using the spectral update condition for feature learning (Yang et al., 2023; Pethick et al., 2025).

Nevertheless, both Muon and Scion have limitations. First, if a weight matrix is split among several devices, it should be gathered on one to compute at every iteration. This leads to higher communication costs. Moreover, Muon optimizes layers 1 and L with AdamW, while Scion proposes to use a different norm for layers 1 and L, leading to a costly hyperparameter search. Lastly, while there has been evidence of learning rate transfer when scaling models in width in (Pethick et al., 2025; Bernstein, 2025), it is not clear if there is learning transfer when scaling in depth or even more realistically – co-scale in depth, width and the number of tokens (depth-width-token co-scaling).

The latter scaling is known as the Chinchilla scaling law (Hoffmann et al., 2022). This is a widely used approach to scale the number of tokens during the pre-training proportional to the number of the parameters in the model. The transfer of learning rate when scaling the number of tokens has been a long-standing challenge with a recent attempt in (Bjorck et al., 2025), which gave a rule-of-thumb to transfer the learning rate across token horizon. Learning rate transfer during the depth-width-token co-scaling is an even more challenging open problem (Everett et al., 2024).

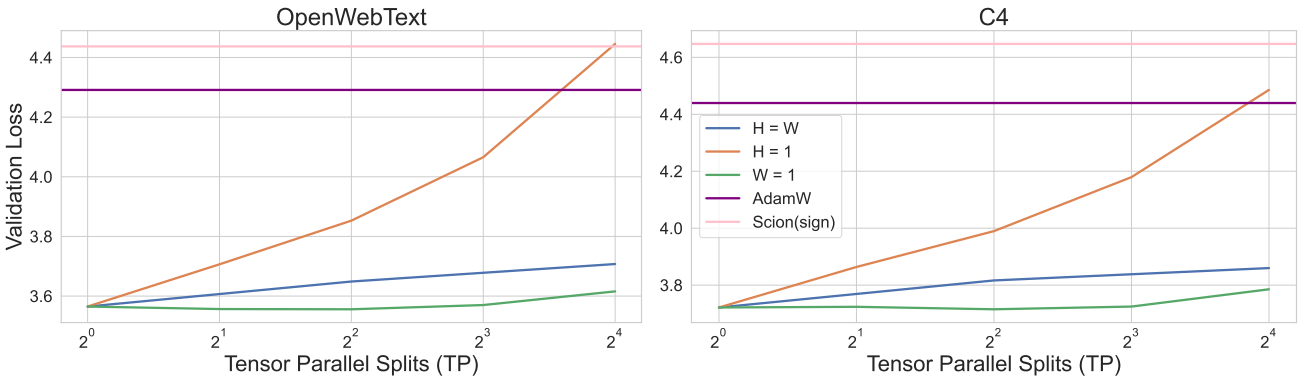

Figure 1. **MuonSBW is more parallelizable than MuonS with comparable performance.** Here, for two datasets - OpenWebText (Gokaslan et al., 2019) and C4 (Raffel et al., 2019) – we investigate for 124M nanoGPT model (Karpathy, 2022) with 1x Chinchilla scaling (Hoffmann et al., 2022) the influence of splitting the momentum gradient matrix introduced in Section 3 into smaller blocks of equal size before the orthogonalization in three ways: i) square blocks (H = W); ii) column splits (H = 1); iii) row splits (W = 1). Increasing the number of blocks initially *improves* loss (row split, TP = 4), and eventually leads to worsening of the loss, which might be explained by the increasing spectral norm of the weights as we discuss in Appendix E.

In this work, we empirically show how we can attempt to solve all of the aforementioned limitations, by combining the scaling update rule suggested by Muon and Scion together with the spectral norm constraint on all layers and block-wise orthogonalization. We name these approaches MuonS and MuonSBW respectively.

1.1. Our Contributions

Our contributions are as follows.

- **MuonSBW.** In Section 4 and Section 5.2, we propose MuonSBW, a more parallelizable version of MuonS, and show in Figure 1 that it is stable in validation loss up to 16x tensor parallelism when training on OpenWebText as well as C4. Moreover, we see that the 4x parallelism slightly outperforms the MuonS baseline discussed in Section 5.1 in more detail.

- **Joint scaling transfer.** Next, in Section 5.3, we show that by only relying on the scaling of updates which are known to enable feature learning and allow for learning rate transfer only in width, we observe that surprisingly learning rate transfers when scaling width, depth of the model and the number of tokens simultaneously. We also confirm the finding on the C4 dataset in Appendix G.1.

- **Static weight decay.** We further investigate in Section 5.4 the influence of weight decay for MuonSBW and in Appendix D for other optimizers. We see that for 5x Chinchilla and bigger models, higher weight decay consistently outperforms lower weight decay values after around 80% of the training run, while being worse than other weight decay values before that.

- **Dynamic weight decay.** We attribute the aforementioned behavior to a lower spectral norm as discussed in Figure 5 and Appendix E. In Section 5.6, we further propose weight decay schedules to improve the performance.

2. Related Work

Throughout this paper, we are interested in solving the unconstrained optimization problem

$$\min_{X \in \mathcal{X}} F(X) \tag{1}$$

for a non-convex $F : \mathcal{X} \to \mathbb{R}$. During the training of a neural network, we search for weights X that solve (1). The compositional structure of neural networks with layers of matrices has spurred research on the non-Euclidean norms used during the optimization (Yang et al., 2023; Jordan et al., 2024; Bernstein & Newhouse, 2024; Pethick et al., 2025; Carlson et al., 2015b;a; 2016; Large et al., 2024), which we discuss in more detail in the following. By minimizing quadratic upper bound at the current iterate X_t we can solve (1) by the *steepest descent* update for general norms (Nesterov, 2010; Mądry, 2015; Carlson et al., 2016) (for more details see Appendix A)

$$\boxed{X_{t+1} = X_t - \frac{\|\nabla F(X_t)\|_*}{L} \big(\nabla F(X_t)\big)^{\#}_{\|\cdot\|},} \qquad \big(\nabla F(X_t)\big)^{\#}_{\|\cdot\|} \in \arg\max_{\|H\|=1} \langle \nabla F(X_t), H \rangle. \tag{2}$$

Stochastic Spectral Descent (SSD). In (Carlson et al., 2015b;a; 2016), the authors applied gradient descent for general norms to neural networks. Concretely, they proposed

Table 1. **Optimizers difference.** Muon (Jordan et al., 2024) and Scion (Pethick et al., 2025) use SSD for some of the layers of the network, while for others they suggest to use an equivalent of AdamW (for Muon) or Signum (Bernstein et al., 2018) (for Scion). We suggest using spectral descent for all layers, which we abbreviate as MuonS together with its block-wise version as MuonSBW, which effectively requires only one learning rate to tune.

Method	First layer	Middle layers	Last layer
Scion	Signum	Spectral	Signum
Muon	AdamW	Spectral	AdamW
MuonSBW (ours)	Spectral	Spectral	Spectral

to replace the gradient oracle $\nabla F(\cdot)$ with the stochastic gradient oracle $\nabla f(\cdot, \xi)$ and perform the update (2) with respect to the matrix norm, namely the spectral norm, since for the neural networks the iterates are structured as weight matrices. This led to significant speed-ups when training smaller feedforward and convolutional neural networks as well as Restricted Boltzmann Machines (RBMs) and other probabilistic models.

Muon. When using the spectral norm (largest singular value of a matrix) $\sigma_{\max}(\cdot)$ in (2), the sharp operator $(X)^{\#}_{\sigma_{\max}(\cdot)}$ is a semi-orthogonal matrix closest to X, which can be computed using singular value decomposition (SVD). In (Jordan et al., 2024), however, the authors replaced SVD with Newton-Schulz iteration (NS), which is more efficient on modern GPUs. Moreover, they incorporated an additional momentum term in SSD. This allowed for speed-ups compared with AdamW during both nanoGPT (Karpathy, 2022) and CIFAR-10 (Krizhevsky & Hinton, 2009) speed-runs. However, their algorithm does not apply SSD for each layer. Instead, it still uses AdamW for the first and last layers of a neural network (see Table 1).

Distributed Shampoo As discussed in (Bernstein & Newhouse, 2024), Shampoo (Gupta et al., 2018) without accumulation can be viewed as SSD. It is therefore interesting to understand approaches for distributed optimization with it. In (Shi et al., 2023a), the authors propose in Section 4.2 to allow for tensor parallelism by strategy called "blocking", which effectively applies Shampoo on each of the blocks. This served as an inspiration for our approach that we introduce in Section 4 and investigate in more detail in Section 5.2.

Scion. The Stochastic Conditional Gradient with Operator Norms (Scion) optimizer introduced in (Pethick et al., 2025) is a concurrent work based on the Stochastic Conditional Gradient method (SCG) that provides a control over the norm in each layer of a neural network. The authors proposed using the $\|\cdot\|_\infty$ norm in SCG for the first and last layers, effectively applying the Signum optimizer for them. For the other layers, they follow propose to use spectral norm as in Muon and SSD. Learning rates for these two different norms (spectral and $\|\cdot\|_\infty$) should be tuned separately. Because our first aim is to speed up the hyperparameter search by simplifying the algorithm, we instead use SSD with spectral norm constraint for all layers (see Table 1).

Moonlight. In (Liu et al., 2025), the authors have trained big Mixture-of-Experts models (MoE) with 3B and 16B parameters for longer than 17x Chinchilla using both Muon and AdamW. Muon improves the Pareto frontier achieving a lower loss with much less training FLOPs. They further compare it with AdamW and show that Muon is approximately 2x more computationally efficient compared to AdamW. On this big scale, they show, among other findings, that a higher weight decay of 0.1 for most of the training run performs worse for Muon while starting to outperform less and no weight decay at the end of the training. Because this scale is not out of reach for many academic labs, in Section 5.4 we investigate if this phenomenon is happening at a smaller scale, for models with up to 758M parameters and with 1x and 5x Chinchilla scaling. Moreover, in Figure 5 and Appendix E, we show that better generalization occurs when spectral norm is constrained more tightly.

3. Muon

In this section we introduce the Muon algorithm with weight decay and its main properties. We discuss it in more detail in Appendix B.

Norm. In (Bernstein & Newhouse, 2024), the authors show that algorithms, such as SGD, Adam, and Shampoo (Gupta et al., 2018) are all related to each other via the steepest descent problem (8) discussed in Appendix A by the choice of the norm $||\cdot|| - l_2, l_\infty$, and $\sigma_{\max}(\cdot)$ – respectively. Muon, similarly to Shampoo, uses on the steepest descent updates with respect to the spectral norm.

Algorithm. For $\eta_t, \lambda \geq 0$ and a neural network with L layers, Muon is orthogonalizing updates to weight matrices W_t^l for layers $l \in \{2, \ldots, L-1\}$ at each time step t to compute the steepest descent direction $(X)^{\#}_{||\cdot||_{\text{RMS}\to\text{RMS}}} = \sqrt{\frac{d^l_{\text{out}}}{d^l_{\text{in}}}} Ortho(G_t^l)$ with respect to the RMS-to-RMS operator norm (more details are in Appendix B):

$$W_{t+1}^l = W_t^l - \eta_t \Big(\sqrt{\frac{d^l_{\text{out}}}{d^l_{\text{in}}}} Ortho(G_t^l) + \lambda W_t^l \Big), \quad (3)$$

where $G_t^l \in \mathbb{R}^{d^l_{\text{out}} \times d^l_{\text{in}}}$ is the momentum stochastic gradient

$$G_t^l = (1-\alpha_t) G_{t-1}^l + \alpha_t \nabla f(W_t^l, \xi_t). \quad (4)$$

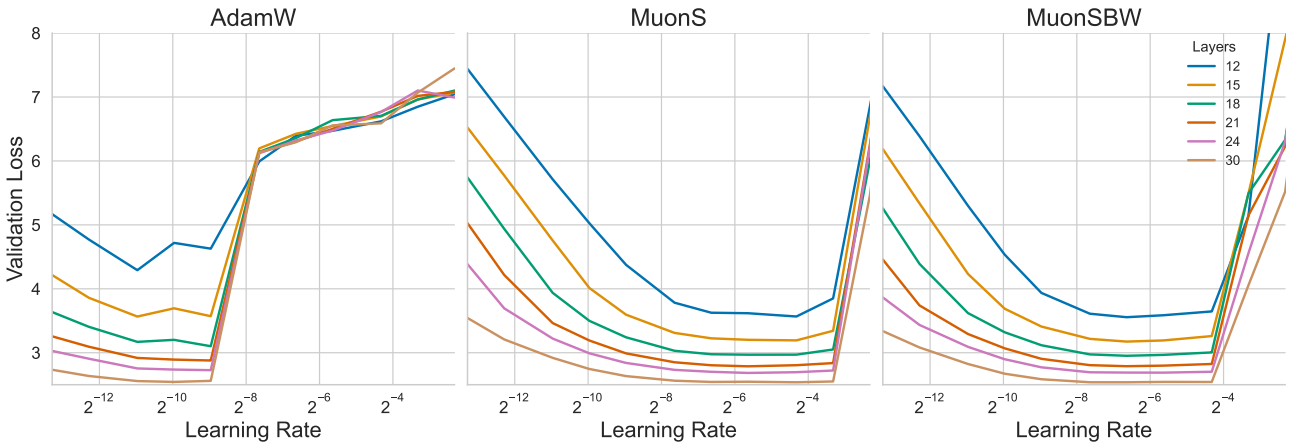

Figure 2. **MuonS and MuonSBW enjoy learning rate transfer during depth-width-token co-scaling.** We are training nanoGPT on OpenWebText with 1x Chinchilla scaling. For both MuonS and MuonSBW learning rate transfers well, unlike for AdamW. Moreover, MuonSBW enjoys a better transfer: the same learning rate of 0.01 is the best across all sizes (see Table 2), staying the best also for another dataset, C4, in Figure 20 in Appendix. We report optimal learning rate values together with achieved validation loss in Table 2.

$Ortho(G_t^l)$ returns the closest semi-orthogonal matrix:

$$Ortho(G_t^l) = \text{argmin}_{O \in \mathcal{O}_{m_l \times n_l}} \|O - G_t^l\|_F, \quad (5)$$

where one minimizes over semi-orthogonal matrices.

$$\mathcal{O}_{m\times n} := \{A \in \mathbb{R}^{m\times n} \mid AA^\top = I_{m\times m} \text{ or } A^\top A = I_{n\times n}\}.$$

4. Block-wise Orthogonalization

Inspired by (Shi et al., 2023b), we propose to i) split the momentum gradient matrix before orthogonalization into tensor parallel splits (TP) row-, column-, or block-wise, doing so in sub-matrices of equal dimensions; ii) compute with NS orthogonalized sub-matrices; iii) concatenate later it as one matrix. The influence of this varying granularity can be seen in Figure 1, showing how we interpolate between the two modes of orthogonalizing the whole matrix on the left (TP = 2^0) and orthogonalizing an increasing number of sub-blocks separately until TP = 2^4.

5. Experiments

We train the original nanoGPT (Karpathy, 2022), without changing its initialization, on OpenWebText (Gokaslan et al., 2019) and C4 (Raffel et al., 2019) datasets. In all experiments, we increase the size of the model by increasing the number of its layers (*depth scaling*); set the number of attention heads to be equal to the number of layers, while increasing the embedding dimension proportionally by the factor 64 (*width scaling*); moreover, for each experiment we use 1x or 5x Chinchilla scaling for the number of tokens depending on the setting (*token number scaling*). In the experiments, we train and analyze models of different sizes, from 124M up to 1.43B parameters, more precisely (we include the number of layers in the brackets): 124M (12), 215M (15), 345M (18), 524M (21), 758M (24), 1.43B (30). For more details, see Appendix H.

5.1. Orthogonalizing All Layers

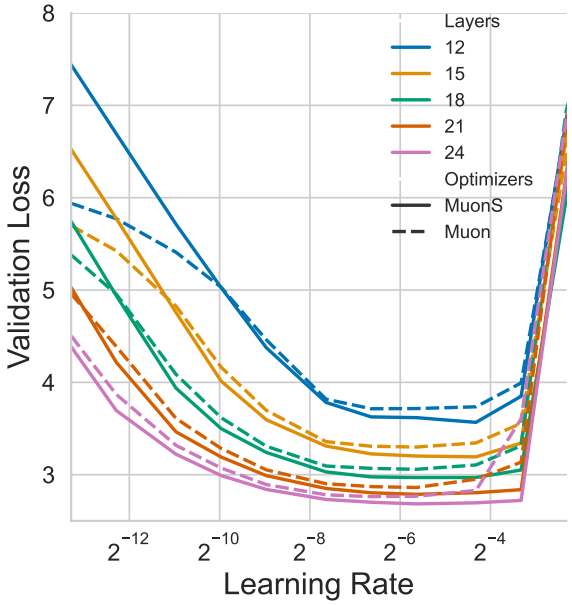

Figure 3. **By tuning only one hyperparameter, MuonS is better than Muon.** We train nanoGPT on OpenWebText with 1x Chinchilla. We increase the size of the model by simultaneously increasing its depth and width, represented by the number of layers in the legend. We can also observe that for both algorithms there is a transfer of learning rate.

Table 2. **For MuonS and MuonSBW we observe learning rate transfer during depth-width-token co-scaling.** We train nanoGPT models on OpenWebText with 1x Chinchilla scaling. We plot validation losses for all tested learning rate values in Figure 4.

Size	AdamW		MuonS		MuonSBW	
	LR	Loss	LR	Loss	LR	Loss
124M	5.0e-04	4.2912	0.05	3.5668	0.01	**3.5557**
215M	5.0e-04	3.5657	0.05	3.1932	0.01	**3.1743**
345M	0.002	3.1019	0.02	2.9692	0.01	**2.9522**
524M	0.002	2.8785	0.02	**2.7876**	0.01	2.7892
758M	0.002	2.7271	0.02	**2.6839**	0.02	2.6885
1.43B	0.001	2.5435	0.05	2.5377	0.01	**2.5377**

In the original Muon implementation, AdamW was chosen for the first and last layers. However, to speed up hyperparameter training, stabilize the training, and simplify the optimizer, we use the spectral norm constraint for all layers, which we name MuonS. When optimizing one learning rate and keeping the rest of the hyperparameters fixed (more details are in Appendix H), it outperforms Muon and AdamW for all model sizes, as can be seen in Figure 3 and Figure 2. Tuning additional hyperparameters can, of course, lead to better performance, as we demonstrate in Appendix C.

5.2. Analyzing Block-wise Orthogonalization

One of the downsides of the Muon optimizer is the need to gather the update matrix G_t^l on one device, which incurs additional communication costs during tensor parallel training. For a naive implementation, it can save communication cost during Allgather operation of $d_{\text{out}} \times d_{\text{in}}$ orthogonalizing the update matrix for each layer. To understand the influence of block-wise orthogonalization on the validation loss achieved by the model, we compare in Figure 1 several different ways of splitting (sharding) the update matrix G_t^l. We find that doing row-wise orthogonalization even improves the loss at 4x tensor parallelism and stays stable in validation loss up to 16x tensor parallelism. Degradation in validation loss can be attributed to the increase of the spectral norm as we discuss in Appendix 5.2. As for row-wise 4x tensor parallelism we observed a better performance than the baseline MuonS, we use it as our default setting for MuonSBW in the rest of the experiments. In the next Section, we show the transfer of learning rate during depth-width-token co-scaling.

5.3. Learning Rate Transfer

Next, in Figure 2, we increase the number of layers and thus the size of the network and observe that, unlike AdamW, our suggested optimizers MuonS and MuonSBW, together with the block-wise versions, transfer well. We also report the best validation losses and the learning rates achieving it in Table 2. Surprisingly, for MuonSBW the same learning rate 0.01 is the best for all model sizes, but one, with 24 layers. However, there the learning rate 0.01 is second best and achieves a loss of 2.6896, close to the best.

5.4. Influence of Static Weight Decay

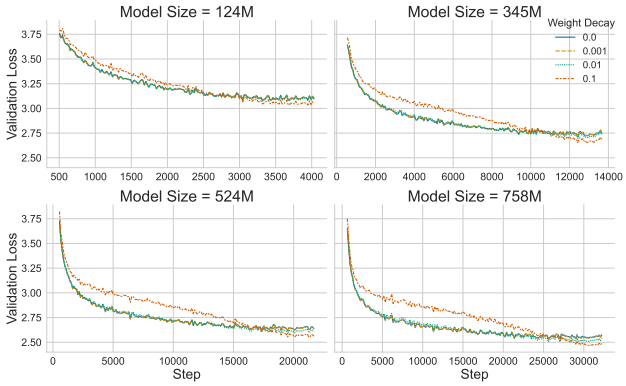

Figure 4. **Higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** Here, for MuonSBW, we vary weight decay for 5x Chinchilla. Similar observation was already made in (Liu et al., 2025), however only for one training run at a much larger scale and only for the MoE architecture. In Appendix D, we observe similar trend for 1x Chinchilla scaling and in Appendix G.2 - for models trained on C4.

Further, we investigate the influence of weight decay by training models with 5x Chinchilla scaling. We can observe in Figure 4 that throughout most of the training for all model sizes, the higher weight decay 0.1 has a higher validation loss than the lower weight decay values. In the end, roughly at the last 80% of the training (more details are in Appendix D), the value of 0.1 results in significantly lower validation loss. This can be attributed to the spectral norm, as we discuss in the next section.

5.5. Investigating Spectral Norms

To better understand the reason behind the suddenly better performance of the higher weight decay at the end of the training, we analyze the spectral norm of the 124M model for the LM head. We observe in Figure 5 the following: i) (*on the left*) a higher weight decay value of 0.1 has a significantly lower spectral norm, in line with the optimization constrained to the spectral norm-ball that it induces, as we discuss in Appendix B.2; ii) (*in the middle*) increasing the number of splits of the gradient update matrix for MuonSBW leads to a higher spectral norm that indicates a gradually worse approximation of the orthogonalization due to a higher number of splits, which still works well in practice as we can see in Figure 1; iii) (*on the right*) MuonSBW has the lowest spectral norm, likely due to the spectral norm being enforced on the first and last layers additionally, com-

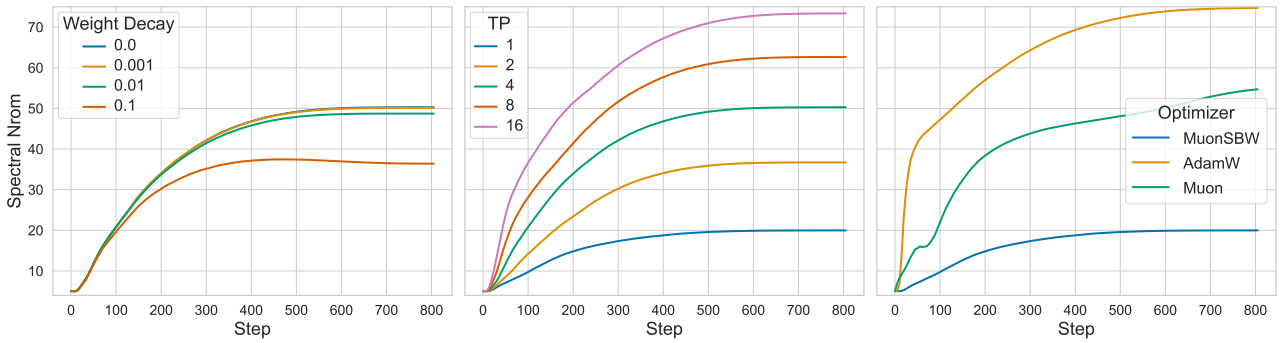

Figure 5. **Higher weight decay of leads to a significantly lower spectral norm.** For the 124M model, we compute the spectral norm of the last layer, LM head, while varying weight decay, number of tensor parallel splits for MuonSBW, and optimizer. We see that a higher weight decay of 0.1, on the left, leads to a significantly lower spectral norm, which might explain its better generalization properties.

pared to Muon, which uses AdamW for them. In addition, we computed the spectral norm for each of the layers: 124M model has 12 layers or transformer blocks, each containing 4 weight matrices: 2 in MLP and 2 in self-attention. For each layer/block, we take the maximal spectral norm across 4 weight matrices, computed with singular value decomposition (SVD) for higher precision, and report it across training steps. We discuss it further in Appendix E.

5.6. Dynamic Weight Decay

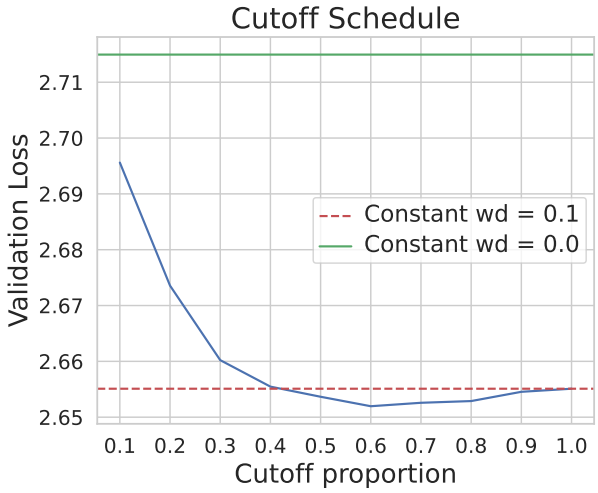

Figure 6. **Clipping weight decay at the end improves the validation loss.** For the 345M model and 5x Chinchilla scaling, clipping at the 80% of the training improves the validation loss. See Appendix D for more experiments and details.

We observed in Section 5.4 that the weight decay of 0.1 outperforms at the end of training, likely due to the tighter constraints on the spectral norm throughout the training, as we observed in the previous section and Appendix E. Here, we investigate whether the spectral norm constraint should be relaxed by decreasing weight decay towards the end of the training. For this, we try three weight decay schedules:

Cutoff: $w_c(t) = 0.1 \cdot \chi_{t \leq t_{\max} \cdot t_c}$, where weight decay is 0.1 up until the cutoff proportion $t_c \in [0, 1]$ and 0 after that;

Polynomial, $w_p(t) = 0.1 \cdot \frac{t^k}{t^k_{\max}}$;

Inverse Polynomial, $w_p(t) = 0.1 \cdot (1 - \frac{t^k}{t^k_{\max}})$.

For the 124M model, we show in Figure 6 that decreasing weight decay towards the end of the training reduces validation loss - both using Cutoff and Inverse Polynomial schedules. Thus, we take the simpler schedule, Cutoff, and train a larger 345M model, with the results in Figure 6. We confirm that the Cutoff schedule achieves a lower validation loss than the baseline. More details are in the Appendix D.

6. Discussion

Conclusion. For the Muon optimizer, we propose using the spectral norm constraint for all layers to speed up the hyperparameter search, perform block-wise orthogonalization to improve communication efficiency when using tensor parallelism, and observe learning rate transfer across model sizes when co-scaling depth, width, and the number of tokens. Additionally, we investigate in more detail the weight decay and CBS of the proposed optimizers.

Limitations. While we show that block-wise orthogonalization works, achieving the best results with row-wise sharding, we do not implement it in Megatron-LM or a similar framework, which would be important for a better adoption, and leave it for future work. Moreover, we investigate the learning rate transfer and behavior of block-wise orthogonalization and weight decay empirically and leave theoretical understanding for future work as well. Lastly, due to the high computational cost, we provide results for one seed.

References

Arora, S., Li, Z., and Panigrahi, A. Understanding gradient descent on the edge of stability in deep learning. In *ICML*, 2022.

Bernstein, J. Deriving muon, 2025. URL https://jeremybernste.in/writing/deriving-muon.

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

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. signSGD: Compressed optimisation for non-convex problems. In *ICML*, 2018.

Bjorck, J., Benhaim, A., Chaudhary, V., Wei, F., and Song, X. Scaling optimal LR across token horizons. In *ICLR*, 2025.

Carlson, D., Cevher, V., and Carin, L. Stochastic spectral descent for restricted boltzmann machines. In *AISTATS*, 2015a.

Carlson, D., Hsieh, Y.-P., Collins, E., Carin, L., and Cevher, V. Stochastic spectral descent for discrete graphical models. *Selected Topics in Signal Processing*, 2016.

Carlson, D. E., Collins, E., Hsieh, Y.-P., Carin, L., and Cevher, V. Preconditioned spectral descent for deep learning. In *NeurIPS*, 2015b.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Cohen, J., Kaur, S., Li, Y., Kolter, J. Z., and Talwalkar, A. Gradient descent on neural networks typically occurs at the edge of stability. In *ICLR*, 2021.

Everett, K., Xiao, L., Wortsman, M., Alemi, A. A., Novak, R., Liu, P. J., Gur, I., Sohl-Dickstein, J., Kaelbling, L. P., Lee, J., and Pennington, J. Scaling exponents across parameterizations and optimizers. In *ICML*, 2024.

for AI, T. A. I. C4 corpus. https://huggingface.co/datasets/allenai/c4, 2019.

Gokaslan, A., Cohen, V., Pavlick, E., and Tellex, S. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Gupta, V., Koren, T., and Singer, Y. Shampoo: Preconditioned stochastic tensor optimization. In *ICML*, 2018.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. An empirical analysis of compute-optimal large language model training. In *NeurIPS*, 2022.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Karpathy, A. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Large, T., Liu, Y., Huh, M., Bahng, H., Isola, P., and Bernstein, J. Scalable optimization in the modular norm. In *NeurIPS*, 2024.

Littwin, E. and Yang, G. Tensor programs ivb: Adaptive optimization in the infinite-width limit. In *ICLR*, 2023.

Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., Chen, Y., Zheng, H., Liu, Y., Liu, S., Yin, B., He, W., Zhu, H., Wang, Y., Wang, J., Dong, M., Zhang, Z., Kang, Y., Zhang, H., Xu, X., Zhang, Y., Wu, Y., Zhou, X., and Yang, Z. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Mądry, P. Lecture 13: Mirror descent. https://people.csail.mit.edu/madry/6S978/files/lecture_13.pdf, 2015. MIT Course 6.S978/18.S997, "Optimization for Theoretical Computer Science".

Mokhtari, A., Hassani, H., and Karbasi, A. Stochastic conditional gradient methods: from convex minimization to submodular maximization. *JMLR*, 2020.

Molybog, I., Albert, P., Chen, M., DeVito, Z., Esiobu, D., Goyal, N., Koura, P. S., Narang, S., Poulton, A., Silva, R., Tang, B., Liskovich, D., Xu, P., Zhang, Y., Kambadur, M., Roller, S., and Zhang, S. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*, 2023.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. Discussion paper, 2010.

Nesterov, Y. *Lectures on Convex Optimization*. Springer Publishing Company, Incorporated, 2nd edition, 2018.

Pethick, T., Xie, W., Antonakopoulos, K., Zhu, Z., Silveti-Falls, A., and Cevher, V. Training deep learning models with norm-constrained lmos. *arXiv preprint arXiv:2502.07529*, 2025.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Shi, H.-J. M., Lee, T.-H., Iwasaki, S., Gallego-Posada, J., Li, Z., Rangadurai, K., Mudigere, D., and Rabbat, M. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023a.

Shi, H.-J. M., Lee, T.-H., Iwasaki, S., Gallego-Posada, J., Li, Z., Rangadurai, K., Mudigere, D., and Rabbat, M. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale, 2023b.

Yang, G. and Hu, E. J. Tensor programs iv: Feature learning in infinite-width neural networks. In *ICML*, 2021.

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., and Gao, J. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. In *NeurIPS*, 2021.

Yang, G., Simon, J. B., and Bernstein, J. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Yang, G., Yu, D., Zhu, C., and Hayou, S. Tensor programs VI: Feature learning in infinite depth neural networks. In *ICLR*, 2024.

Zhang, H., Morwani, D., Vyas, N., Wu, J., Zou, D., Ghai, U., Foster, D., and Kakade, S. M. How does critical batch size scale in pre-training? In *ICLR*, 2025.

Zhang, J., He, T., Sra, S., and Jadbabaie, A. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *ICLR*, 2020.

A. Steepest Descent for General Norms

L-smoothness. Assume that F is differentiable with L-Lipschitz gradient with respect to a general norm $\|\cdot\|$ (that is F is L-smooth with respect to $\|\cdot\|$),

$$\|\nabla F(X) - \nabla F(Y)\|_* \leq L\,\|X - Y\|, \qquad \forall X, Y \in \mathcal{X}, \tag{6}$$

where $\|\cdot\|_*$ is the dual norm

$$\|G\|_* = \sup_{\|X\|\leq 1} \langle G, X\rangle, \tag{7}$$

Then for $\mathcal{X} = \mathbb{R}^n$ by descent lemma (Nesterov, 2018) for all X, Y this implies (and for F convex it is equivalent by (Nesterov, 2018))

$$F(Y) \leq F(X) + \langle \nabla F(X), Y - X\rangle + \frac{L}{2}\,\|Y - X\|^2. \tag{8}$$

Inequality (8) provides a quadratic upper bound (majoriser) of F around X. Minimizing this surrogate at the current iterate X_t allows to solve (1) by the *steepest descent* update for general norms (Nesterov, 2010; Mądry, 2015; Carlson et al., 2016)

$$\boxed{X_{t+1} = X_t - \frac{\|\nabla F(X_t)\|_*}{L}\big(\nabla F(X_t)\big)^{\#}_{\|\cdot\|}} \qquad \big(\nabla F(X_t)\big)^{\#}_{\|\cdot\|} \in \arg\max_{\|H\|=1} \langle \nabla F(X_t), H\rangle. \tag{9}$$

For the Euclidean ℓ_2 norm, $\big(\nabla F(X_t)\big)^{\#}_{\|\cdot\|_2} = \frac{\nabla F(X_t)}{\|\nabla F(X_t)\|_2}$, so the steepest descent step in (2) reduces to gradient descent (GD) with step size $1/L$. The standard GD $X_{t+1} = X_t - \eta_t \nabla F(X_t)$ with an arbitrary η_t is therefore not the steepest descent update unless $\eta_t = 1/L$. Note as well that for the standard GD L-smoothness is a sufficient descent condition, guaranteeing that $F(X_{t+1}) < F(X_t)$, as long as $\eta_t < \frac{2}{L}$.

Neural Networks. It is well known that neural networks do not admit L-smoothness with respect to the Euclidean norm (Cohen et al., 2021; Zhang et al., 2020; Large et al., 2024). On the other hand, while global (or even local) L-smoothness is sufficient for monotone descent, neural network training can succeed without it: full-batch GD stabilizes at $\lambda_{\max} \approx 2/\eta$ – hovering right at or just above the strict upper bound for the sufficient descent condition – yet still converges (Cohen et al., 2021; Arora et al., 2022). In such cases, we can interpret L not as a Lipschitz constant, but as the "sharpness" (Bernstein & Newhouse, 2024) – by decreasing sharpness, we increase the step size.

B. More Details on Muon

B.1. Feature learning.

The spectral norm is motivated by recent work (Yang et al., 2023), where feature learning condition is derived for MLP with weight matrices $W_t^l \in \mathbb{R}^{d^l_{\text{out}} \times d^l_{\text{in}}}$. It requires that for each step t and each layer l the following holds:

$$\sigma_{\max}(W_t^l) = \Theta\Big(\sqrt{\frac{d^l_{\text{in}}}{d^l_{\text{out}}}}\Big),$$

$$\sigma_{\max}(W_{t+1}^l - W_t^l) = \Theta\Big(\sqrt{\frac{d^l_{\text{in}}}{d^l_{\text{out}}}}\Big).$$

It holds for (2) with the norm $||W_t^l||_{\text{RMS}\to\text{RMS}} = \sqrt{\frac{d^l_{\text{in}}}{d^l_{\text{out}}}}\sigma_{\max}(W_t^l)$, $||x||_{\text{RMS}} := \sqrt{\frac{1}{d^l_{\text{in}}}\sum_{i=1}^{d^l_{\text{in}}} x_i^2}$.

B.2. Constrained Optimization via Weight Decay.

By explicitly choosing different norms for each layer, the concurrent work of (Pethick et al., 2025) builds on the Stochastic Conditional Gradient method (SCG), introduced in (Mokhtari et al., 2020). Concretely, in (Pethick et al., 2025), the authors rewrite Muon update (3) by constraining $\eta_t \in (0, 1)$ and $\lambda \in [0, 1]$ as follows

$$W_{t+1}^l = (1 - \eta_t\lambda)W_t^l + \eta_t\,\text{lmo}_{||\cdot||_{\text{RMS}\to\text{RMS}}}(G_t^l). \tag{10}$$

Here, lmo stands for linear minimization oracle defined as

$$\mathrm{lmo}_{\|\cdot\|,r}(X) \in \underset{\{S \in \mathcal{X} \mid \|S\| \leq r\}}{\arg\min} \langle X, S \rangle.$$

and can be expressed via sharp operator: $\mathrm{lmo}_{\|\cdot\|,r}(X) = -r(X)^{\#}_{\|\cdot\|}$.

The formulation in (10) makes it clear that when $\eta_t \in (0, 1)$ and $\lambda = 1$, we recover SCG, which minimizes our main objective (1) in the norm-ball of radius $\mathcal{D} \coloneqq \{X \in \mathcal{X} \mid \|X\| \leq r\}$.

C. Tuning more hyperparameters

In this paper, we focus on MuonS and its parallelizable version, MuonSBW, that require tuning only one hyperparameter. However, in this section, we want to understand the performance of optimizers when tuning more hyperparameters. As this requires expensive testing of all combinations of hyperparameters, we focus on varying two hyperparameters for 124M nanoGPT model with 1x Chinchilla scaling. For this, we train it on OpenWebText.

Muon optimizer uses two optimizers depending on the layer: for the first, last layers, and 1D tensors AdamW is used, while for all others – optimizers based on SSD with momentum and NS iteration. By tuning a separate learning rate for the AdamW-optimized tensors for Muon optimizer in Table 4 and our proposed MuonS (AdamW is used only for 1D tensors) in Table 5, we see that we can achieve a better loss in both cases. While the best performance is achieved with the Muon in this full sweep, note in the case of MuonS that the influence of the second learning rate is not as strong as it is used only for 1D tensors – a property we would like to have in the optimizer that requires tuning only one hyperparameter. For completeness, in Table 6 we have additionally analyzed the performance of MuonS with normalized momentum SGD for 1D tensors. Because performance in validation loss is worse than that of MuonS when using AdamW for 1D tensors in Table 5.

Table 3. **Optimal learning rates and validation losses for MuonS and Scion optimizers.** Here, we train nanoGPT models on OpenWebText. Best and second-best validation losses per model size are highlighted.

Layers	MuonS		Scion	
	LR	Loss	LR	Loss
12	0.05	3.5668	0.05	**3.4342**
15	0.05	3.1932	0.05	**3.1311**
18	0.02	2.9692	0.02	**2.9305**
21	0.02	2.7876	0.02	**2.7798**
24	0.02	2.6839	0.02	**2.6826**
30	0.05	2.5377	0.02	**2.5214**

Next, in Figure 7 and Table 3 we observe that following the optimizer suggested in Scion (Pethick et al., 2025) that uses a different norm constraint for the input and output layers can also improve performance. Concretely, in Scion (Pethick et al., 2025), the authors propose to enforce the ℓ_∞ norm for the input and output layers, while keeping the spectral norm for the rest of the layers. Thus, we separately tune the learning rate for the spectral norm-constrained layers and ℓ_∞ norm-constrained ones.

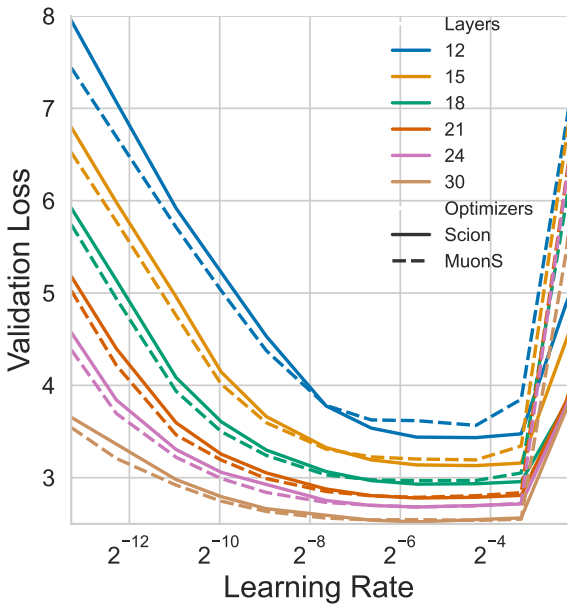

Figure 7. **By tuning additional hyperparameter, Scion optimizer can outperform MuonS.** Here, we show that by tuning an additional hyperparameter for first and last layers, we can achieve a better performance than MuonS. This becomes less evident for larger models. See Table 3 for the optimal learning rate and validation loss values.

Because it has been shown in Scion (Pethick et al., 2025) that such an optimizer enjoys the learning rate transfer when scaling the width, we are interested in both tuning two hyperparameters and testing the learning rate transfer when doing depth-width-token co-scaling, which was the case for Muon, MuonS and MuonSBW (see Section 5.3 and Figure 3), with MuonSBW and MuonS having more consistent behavior of the validation loss when varying learning rate (optimal learning rate is the same for the smallest and the largest models) during such co-scaling. For Scion, we observe in Figure 7 that up-scaling the learning rate by factor 10 for the input and output layers leads to better performance in validation loss and learning rate transfer during depth-width-token co-scaling. We see, however, that MuonS has more consistent behavior of the validation loss and the difference in the achieved validation loss decreases with model scale. This might imply that tuning this additional hyperparameter is less relevant at bigger model scales.

Table 4. Muon validation losses for 124M nanoGPT model trained on OpenWebText. Here, **A** stands for the learning rate used for tensors optimized with *AdamW* - first and last layers, together with 1D tensors. **M** stands for the learning rate used for all the other tensors optimized with Muon. Here, blue cells denote the best validation loss achieved when using the same learning rate during the optimization (faster search), and green - best validation loss achieved when using different learning rates during the optimization (longer search).

M \ A	1e-04	2e-04	5e-04	1e-03	2e-03	5e-03	1e-02	2e-02	5e-02	1e-01	2e-01	5e-01
1e-04	5.9390	5.7853	5.6063	5.5068	5.4671	5.4352	5.4355	5.4525	5.4670	5.5710	5.8037	7.2642
2e-04	5.9344	5.7719	5.5703	5.4461	5.3390	5.2956	5.3305	5.3804	5.4107	5.5289	5.7673	7.0563
5e-04	5.7813	5.6212	5.4087	5.2653	5.1457	5.0251	5.0733	5.2154	5.2319	5.3614	5.6518	6.6242
1e-03	5.5489	5.3877	5.1773	5.0298	4.8968	4.7512	4.7525	4.9443	4.9166	5.0906	5.5074	6.4771
2e-03	5.1832	5.0178	4.8000	4.6233	4.4614	4.3365	4.3910	4.5600	4.5663	4.7187	5.3250	6.5286
5e-03	4.5376	4.3326	4.0997	3.9685	3.8631	3.8181	3.9197	4.0588	4.0772	4.1451	4.9531	6.7412
1e-02	4.2066	4.0190	3.7967	3.6850	3.6212	3.6203	3.7142	3.8321	3.8488	3.9396	4.5877	6.6117
2e-02	4.0653	3.8872	3.6785	3.5718	3.5327	3.5427	3.6310	3.7162	3.7234	3.8101	4.4429	7.4264
5e-02	3.9382	3.7602	3.5800	3.5127	3.4983	3.5136	3.6266	3.6957	3.7353	3.8357	5.2475	6.6402
1e-01	4.2107	3.9947	3.8332	3.7574	3.7060	3.7826	3.8502	3.8895	3.7817	3.9937	5.0316	8.2968
2e-01	6.6563	6.4079	6.1657	6.0540	6.0091	6.0897	6.2917	6.2566	6.4390	6.4332	6.7816	10.9911
5e-01	7.4857	7.0731	6.8243	6.6584	6.5182	6.6184	6.7522	6.9997	6.9901	7.1735	7.2850	10.9911

Table 5. MuonS validation losses for 124M NanoGPT model trained on OpenWebText. Here, **A** stands for the learning rate used for tensors optimized with *AdamW* - only 1D tensors. **M** stands for the learning rate used for all the other tensors optimized with Muon. Here, blue cells denote the best validation loss achieved when using the same learning rate during the optimization (faster search), and green - best validation loss achieved when using different learning rates during the optimization (longer search).

M \ A	1e-04	2e-04	5e-04	1e-03	2e-03	5e-03	1e-02	2e-02	5e-02	1e-01	2e-01	5e-01
1e-04	7.4393	7.3879	7.2704	7.1551	7.0565	6.9783	6.9340	6.9035	6.8836	6.8830	6.8920	6.9085
2e-04	6.7325	6.6915	6.5970	6.5000	6.4076	6.3227	6.2774	6.2477	6.2320	6.2371	6.2522	6.2887
5e-04	5.7924	5.7705	5.7127	5.6501	5.5853	5.5140	5.4755	5.4517	5.4398	5.4456	5.4601	5.4903
1e-03	5.1101	5.0963	5.0629	5.0259	4.9886	4.9470	4.9251	4.9049	4.8959	4.9077	4.9602	5.0222
2e-03	4.4332	4.4351	4.4122	4.3942	4.3753	4.3543	4.3387	4.3267	4.3178	4.3099	4.3175	4.3879
5e-03	3.8076	3.8017	3.7966	3.7913	3.7880	3.7827	3.7728	3.7644	3.7581	3.7715	3.7981	3.8709
1e-02	3.6474	3.6429	3.6366	3.6465	3.6345	3.6426	3.6259	3.6264	3.6227	3.6148	3.6449	3.7259
2e-02	3.8047	3.7775	3.7681	3.7593	3.7375	3.7167	3.6623	3.6181	3.5622	3.5577	3.5773	3.6933
5e-02	4.8114	4.8270	4.8160	4.7428	4.6636	4.4013	3.9651	3.7402	3.5668	3.5463	3.6068	5.4322
1e-01	5.4238	5.4633	5.6575	5.5914	5.4680	5.2629	4.8798	4.0335	3.8014	3.8496	4.9753	7.1162
2e-01	6.1991	6.2095	6.1799	6.1810	6.1312	5.9375	5.4313	5.2340	4.9602	5.4483	6.8416	10.9911
5e-01	10.9911	10.9911	10.9911	10.9911	10.9911	8.6015	6.6321	6.3084	6.3822	9.0017	10.9911	10.9911

Table 6. MuonS validation losses for 124M nanoGPT model trained on OpenWebText. Here, **S** stands for the learning rate used for tensors optimized with *normalized momentum SGD* - only 1D tensors. **M** stands for the learning rate used for all the other tensors optimized with Muon. Here, blue cells denote the best validation loss achieved when using the same learning rate during the optimization (faster search), and green - best validation loss achieved when using different learning rates during the optimization (longer search).

M \ S	1e-04	2e-04	5e-04	1e-03	2e-03	5e-03	1e-02	2e-02	5e-02	1e-01	2e-01	5e-01
1e-04	7.4963	7.4931	7.4829	7.4684	7.4390	7.3606	7.2610	7.1347	6.9964	6.9355	6.9013	6.8954
2e-04	6.7780	6.7749	6.7684	6.7567	6.7337	6.6738	6.5937	6.4869	6.3559	6.2962	6.2648	6.2522
5e-04	5.8165	5.8144	5.8109	5.8027	5.7872	5.7440	5.6818	5.5949	5.4850	5.4250	5.3693	5.3035
1e-03	5.1273	5.1253	5.1207	5.1127	5.0986	5.0580	5.0080	4.9378	4.8401	4.7620	4.7015	4.6365
2e-03	4.4485	4.4444	4.4424	4.4397	4.4303	4.4018	4.3694	4.3194	4.2404	4.1591	4.0989	4.0658
5e-03	3.8122	3.8069	3.8047	3.8057	3.8024	3.7889	3.7795	3.7649	3.7386	3.7246	3.7150	3.7245
1e-02	3.6490	3.6450	3.6458	3.6449	3.6563	3.6417	3.6248	3.6292	3.6189	3.6148	3.6112	3.6230
2e-02	3.8423	3.7921	3.7862	3.7783	3.7711	3.7764	3.7556	3.7265	3.6867	3.6416	3.6111	3.5986
5e-02	4.8113	4.8326	4.7843	4.8653	4.7846	4.7352	4.5963	4.1807	3.9549	3.8951	3.9050	3.8143
1e-01	5.5294	5.6321	5.5654	5.7361	5.4144	5.3642	5.3437	5.2497	4.7069	4.6061	4.8194	5.6131
2e-01	6.2283	6.2298	5.9690	6.0936	6.1978	5.9463	6.1157	5.7292	5.3737	5.5721	6.3683	7.7045
5e-01	10.9911	10.9911	10.9911	10.9911	10.9911	10.9911	10.9911	9.2089	6.7458	7.5215	10.9911	10.9911

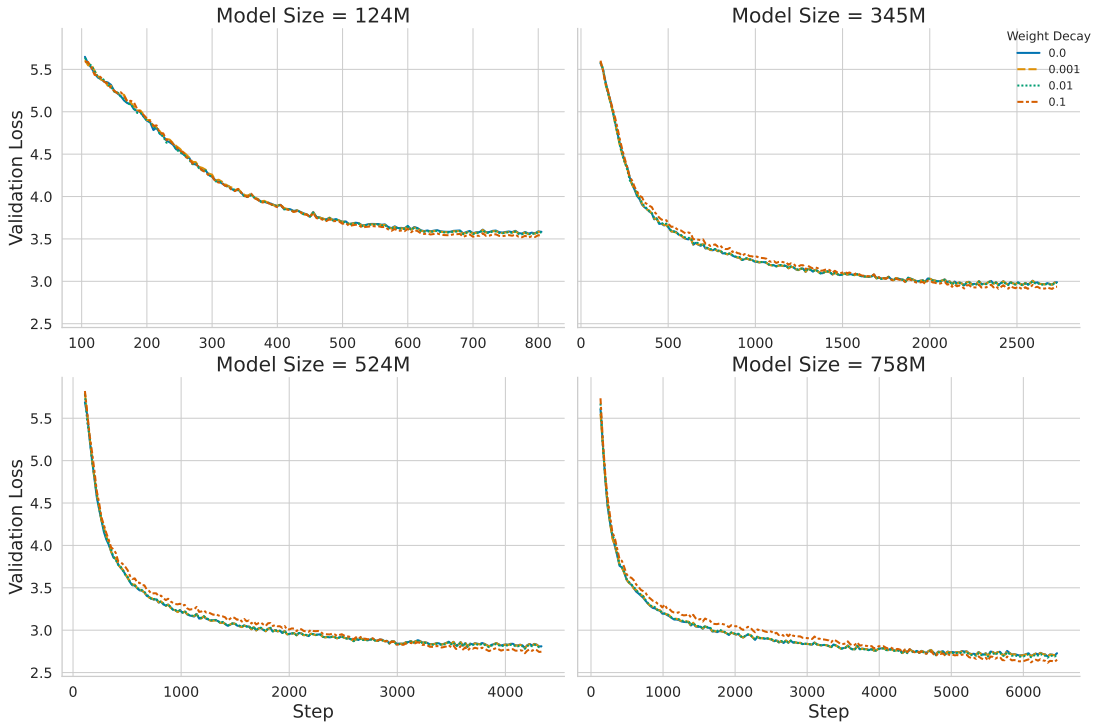

Figure 8. **For MuonSBW, higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** Here, for MuonSBW, we vary weight decay for **1x Chinchilla**. One can see that with 1x Chinchilla scaling it is still visible that at the end of the training higher weight decay of 0.1 consistently outperforms lower weight decay values while being significantly worse before. However, this occurs at the earlier proportion of the training run than using 5x scaling (see Figure 4).

D. Extended Weight Decay Evaluation

In Section 5.4 and Figure 4 for MuonSBW overtrained with 5x Chinchilla scaling, we investigated the influence of weight decay on validation loss. There, the weight decay of 0.1 started to perform better than the lower weight decay values only after 70% of the training run for the 124M model, 78% for the 345M model, 82% for the 524M model, and 84% for the 758M model with a mean of 77%. That is, the bigger the model is and the longer we train (because we co-scale tokens when scaling model size), the later 0.1 weight decay value improves the validation loss.

In this section, we continue training nanoGPT on OpenWebText and do it for additional optimizers, Muon, MuonS, AdamW, as well as for 1x Chinchilla and 5x Chinchilla scaling.

MuonSBW. First, we train using MuonSBW introduced in Section 4, but with fewer tokens, using 1x Chinchilla scaling. In this setting, as we observe in Figure 8 that a similar trend holds – MuonSBW with the weight decay of 0.1 outperforms in validation loss only at the end of the training, however it starts outperforming at the earlier proportion of the training run than with 5x Chinchilla scaling: the weight decay of 0.1 started to perform better than the lower weight decay values only after 32% of the training run for the 124M model, 64% for the 345M model, 68% for the 524M model, and 71% for the 758M model with a mean of 59%.

AdamW. Next, to understand if similar behavior happens for AdamW, we first train it with 5x and then 1x Chinchilla scaling. In both settings, as we observe in Figure 9 for the 5x Chinchilla scaling and in Figure 10 for the 1x Chinchilla scaling, the weight decay of 0.1 outperforms in validation loss only at the end of the training, similar to MuonSBW. However, for both 5x and 1x Chinchilla scaling we observe it only for models of larger sizes – with 524M and 758M parameters, and the difference of performance with higher weight decay is less prominent. Concretely, with the 5x Chinchilla scaling, AdamW with a weight decay of 0.1 started to perform better than the lower weight decay values only after 14% of the training run for the 124M model, 9% for the 345M model, 68% for the 524M model, and 68% for the 758M model with a mean of 40%. For 1x Chinchilla scaling, it starts outperforming already early in the training: after 20% of the training run for the 124M model, 5% for the 345M model, 3% for the 524M model, and 3% for the 758M model with a mean of 8%.

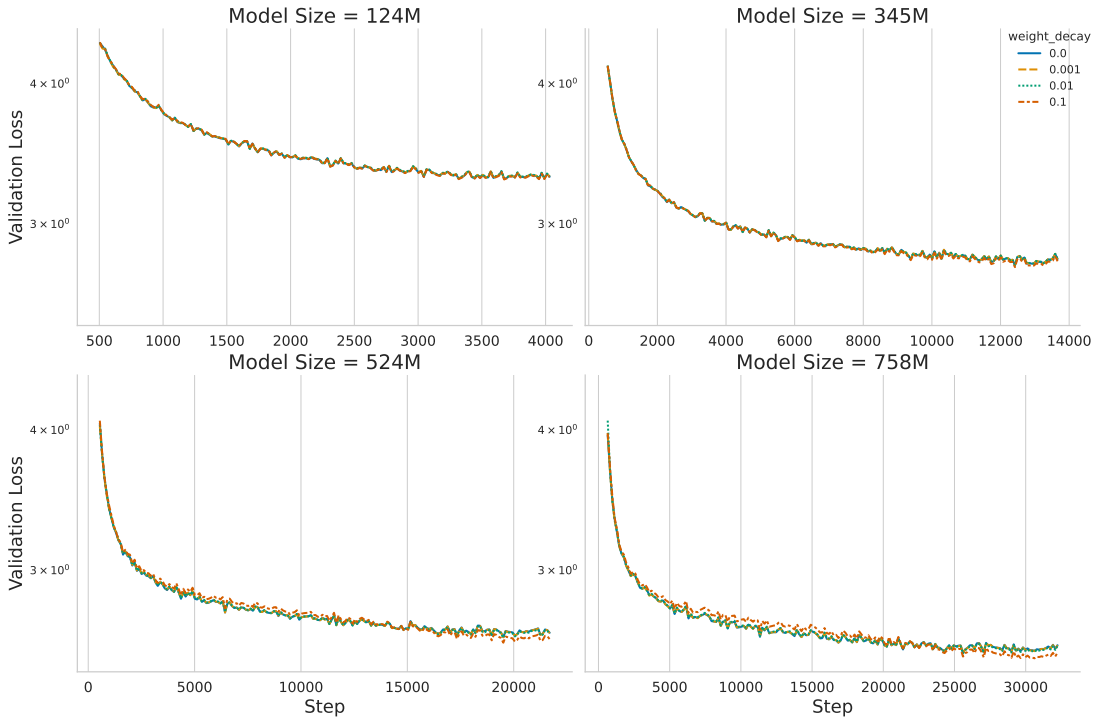

Figure 9. **For AdamW, higher weight decay of** 0.1 **outperforms at the end of the training only for bigger models.** When varying weight decay values for AdamW, for **5x Chinchilla**, we can see that higher weight decay of 0.1 outperforms lower weight decay values at the end of the training only for larger models, 524M and 758M ones.

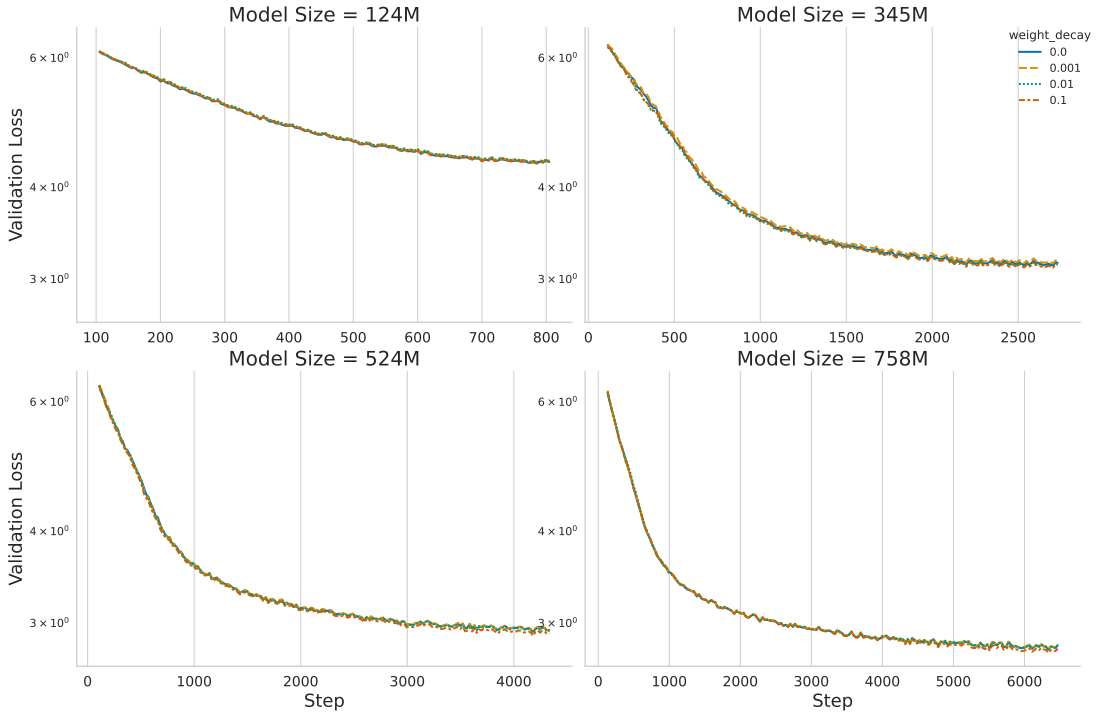

Figure 10. **For AdamW, higher weight decay of** 0.1 **outperforms at the end of the training only for bigger models.** When varying weight decay values for AdamW, for **1x Chinchilla**, we can see that, similarly to **5x Chinchilla** (see Figure 9), higher weight decay of 0.1 outperforms lower weight decay values at the end of the training only for larger models, 524M and 758M ones. The difference becomes less visible however.

MuonS. Similarly, we train with MuonS first with a 5x and then 1x Chinchilla scaling. In both settings, as we observe in Figure 11 for the 5x Chinchilla scaling and in Figure 12 for the 1x Chinchilla scaling, the weight decay of 0.1 outperforms in validation loss only at the end of the training, similar to MuonSBW. Concretely, with the 5x Chinchilla scaling, MuonS with a weight decay of 0.1 started to perform better than the lower weight decay values after 82% of the training run for the 124M model, 89% for the 345M model, 93% for the 524M model, and 99% for the 758M model with a mean of 91%. Thus, this improvement occurs later than for MuonSBW. For 1x Chinchilla scaling, it also occurs later in the training: after 66% of the training run for the 124M model, 79% for the 345M model, 81% for the 524M model, and 84% for the 758M model with a mean of 78%.

Muon. Similarly for Muon, we first train nanoGPT on OpenWebText with 5x and then 1x Chinchilla scaling. In both settings, as we observe in Figure 13 for the 5x Chinchilla scaling and in Figure 14 for the 1x Chinchilla scaling, the weight decay of 0.1 outperforms in validation loss only at the end of the training, similar to MuonSBW. However, for the case of the 5x Chinchilla, we observe a "loss spike" at the end of training. Such "loss spikes" have been observed when training models with AdamW (Molybog et al., 2023), which might be relevant here since Muon uses AdamW to optimize the first and the last layers. In our experiments, with the 5x Chinchilla scaling, Muon with a weight decay of 0.1 started to perform better than the lower weight decay values after 61% of the training run for the 124M model, 77% for the 345M model, 84% for the 524M model, and 89% for the 758M model with a mean of 78%. For 1x Chinchilla scaling, it starts outperforming after 37% of the training run for the 124M model, 53% for the 345M model, 63% for the 524M model, and 68% for the 758M model with a mean of 55%. In both cases, it is similar to MuonSBW.

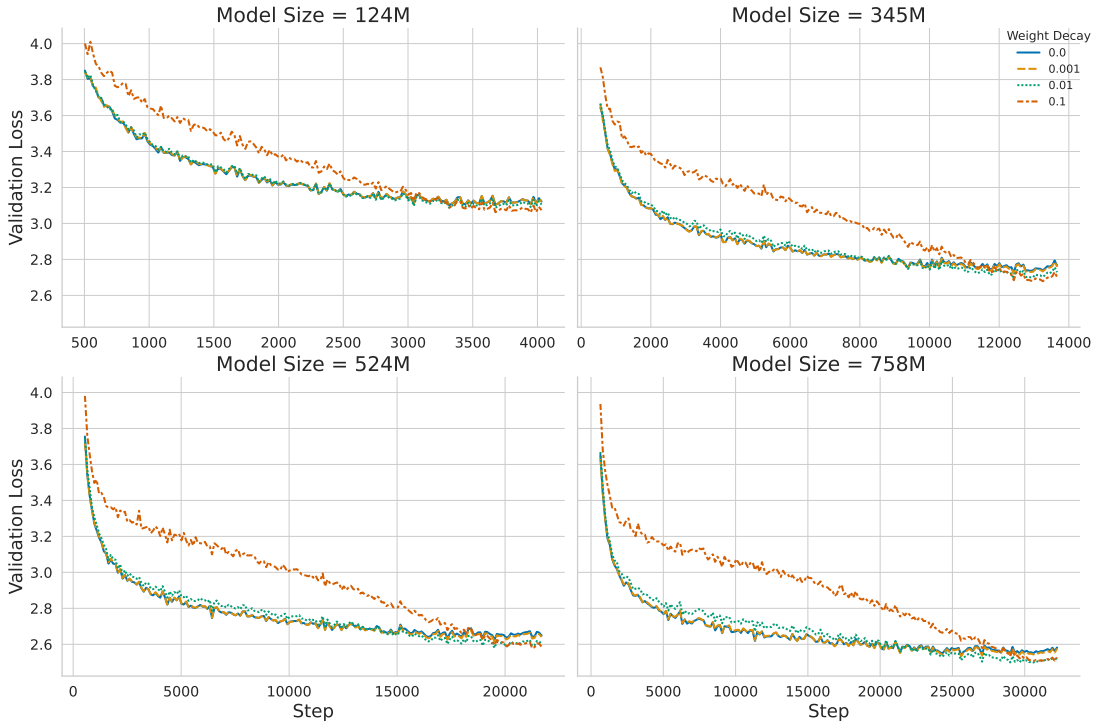

Figure 11. **For MuonS, higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** Here, we vary weight decay for **5x Chinchilla**. In this setting, at the end of the training higher weight decay of 0.1 consistently outperforms lower weight decay values while being significantly worse before. This behavior is similar to using MuonSBW (see Figure 4).

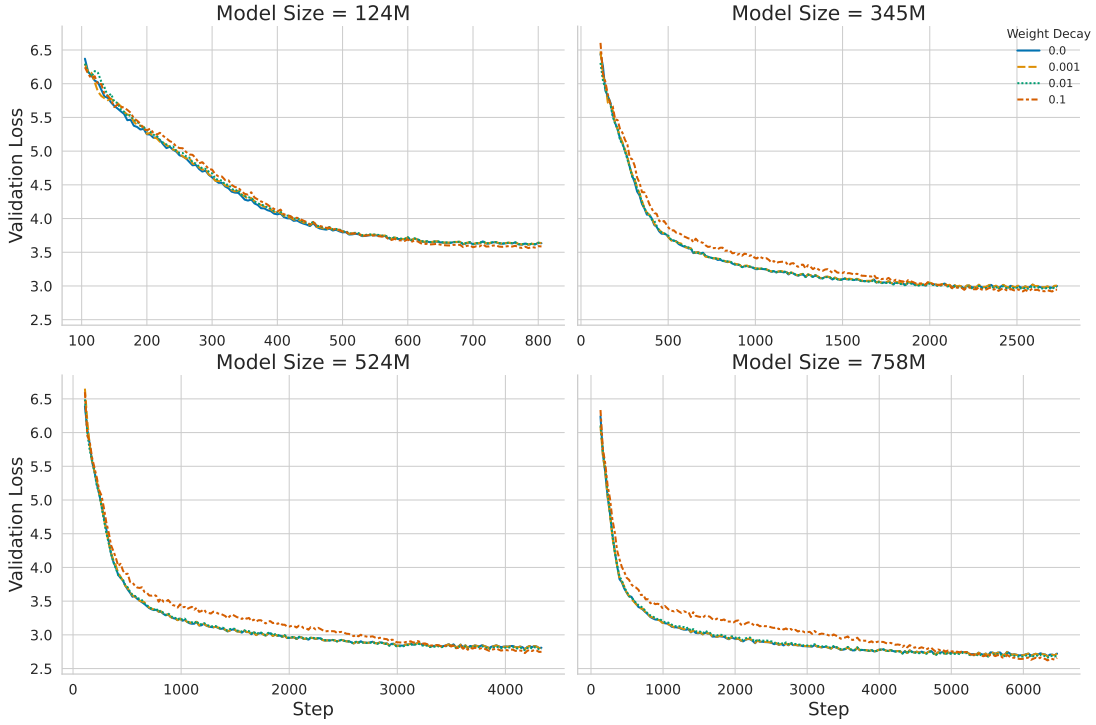

Figure 12. **For MuonS, only at the end of the training higher weight decay of** 0.1 **consistently outperforms.** Here, for MuonS, we vary weight decay for **1x Chinchilla**. In this setting, it is still visible that at the end of the training higher weight decay of 0.1 consistently outperforms lower weight decay values while being significantly worse before. However, this occurs at the earlier proportion of the training run than using 5x scaling (see Figure 11).

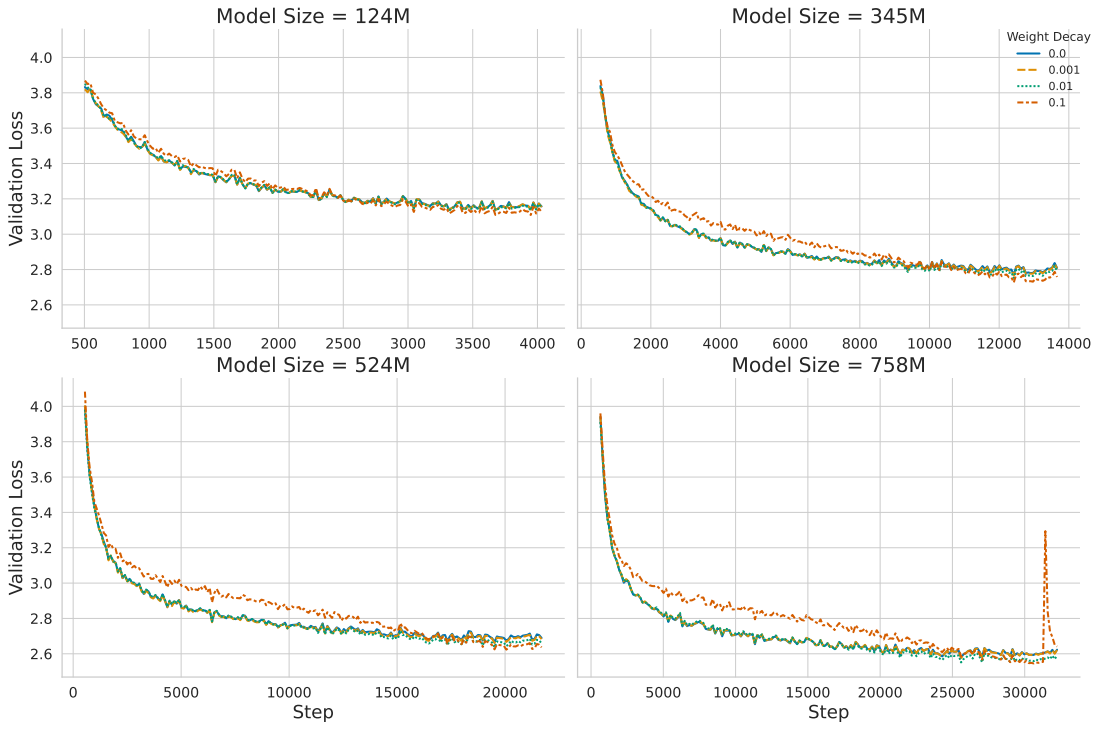

Figure 13. **For Muon, higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** Here, we vary weight decay for **5x Chinchilla**. In this setting, at the end of the training higher weight decay of 0.1 consistently outperforms lower weight decay values while being significantly worse before. This behavior is similar to using MuonSBW (see Figure 4). For the largest model we see a "loss spike" – an undesired artefact, which sometimes occurs for AdamW (Molybog et al., 2023).

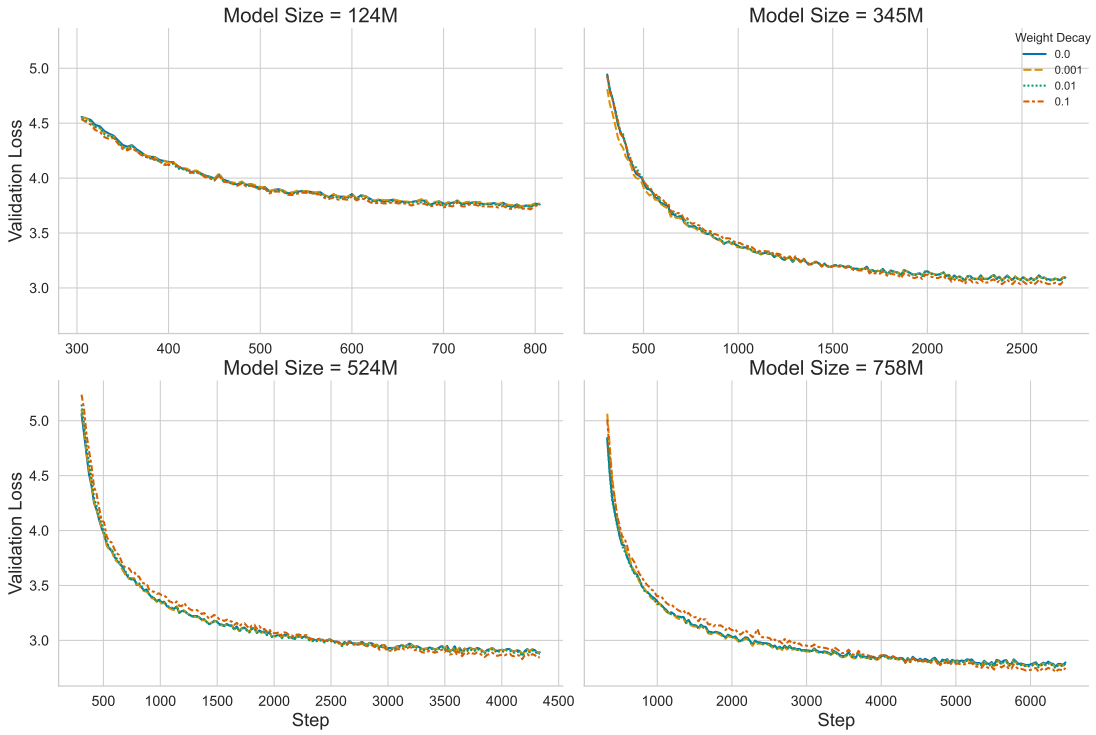

Figure 14. **For Muon, higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** For **1x Chinchilla**, this occurs at the earlier proportion of the training run than using 5x scaling (see Figure 13).

D.1. Weight Decay Schedule

In Section 5.4, we show that for the larger model with 345M parameters, the Cutoff schedule outperforms the baseline (constant weight decay of 0.1) at 80% of the data. Here, for the smaller model with 124M parameters, we do ablation for different schedules introduced in Section 5.4. For it, we observe in Figure 15 that decreasing weight decay with the Inverse Polynomial Schedule leads to similar gains. This, together with the observations in Section 5.4, might indicate that the weight decay becomes less important at the end of the training. We analyze it further from the perspective of a spectral norm in the next section.

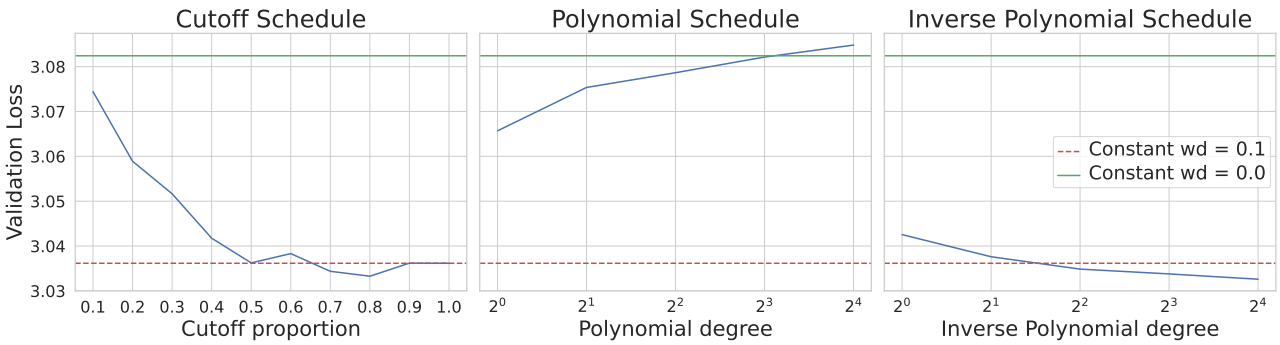

Figure 15. **Clipping weight decay at 80% of the training and Inverse Polynomial Schedule helps.** For MuonSBW, with weight decay clipping (Cutoff Schedule), we can improve compared to the fixed weight decay by turning the weight decay off at the last 80% of the training. We observe similar gains for the Inverse Polynomial Schedule.

E. Investigating Spectral Norms for All Layers

E.1. Influence of the Number of Splits

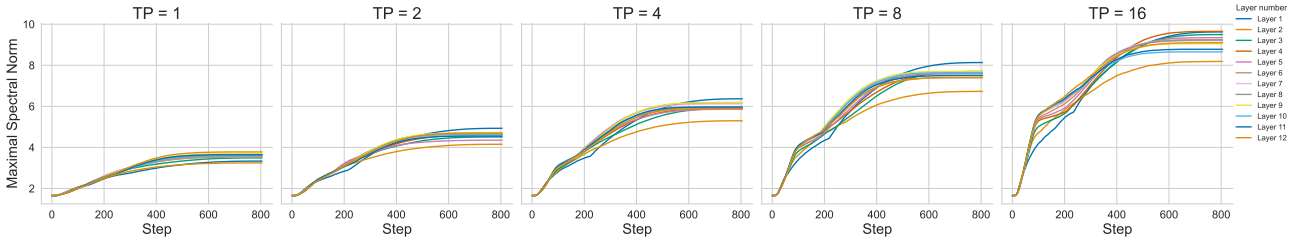

Figure 16. **Increasing the number of tensor parallel splits (TP) in MuonSBW leads to higher spectral norms across all layers.** For the 124M model, we compute the maximal spectral norms across weight matrices in one layer (block of the transformer) varying the number of row-wise splits.

First, in Figure 16, we observe that increasing the number of row-wise tensor parallel splits (TP) for MuonSBW, introduced in Section 4, consistently increases the spectral norms for all layers. We take row-wise splits, as they had the lowest validation loss, close to the MuonS baseline (no block-wise orthogonalization is used) as we could see in Figure 1.

E.2. Influence of the Weight Decay

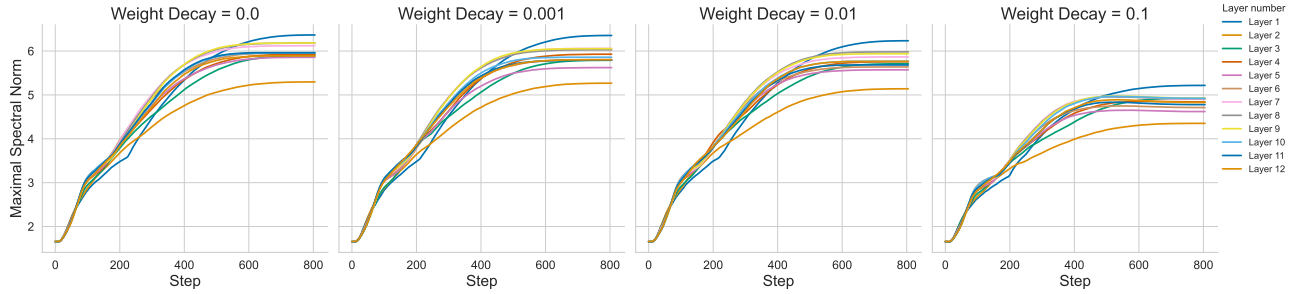

Figure 17. **Increasing the weight decay value in MuonSBW leads to lower spectral norms across all layers.** For the 124M model, we compute the maximal spectral norms across weight matrices in one layer (block of the transformer) varying the number of row-wise splits.

Next, in Figure 17, we see that for MuonSBW we also have a consistent decrease of the spectral norms for all layers when increasing the weight decay value.

E.3. Influence of the Optimizer

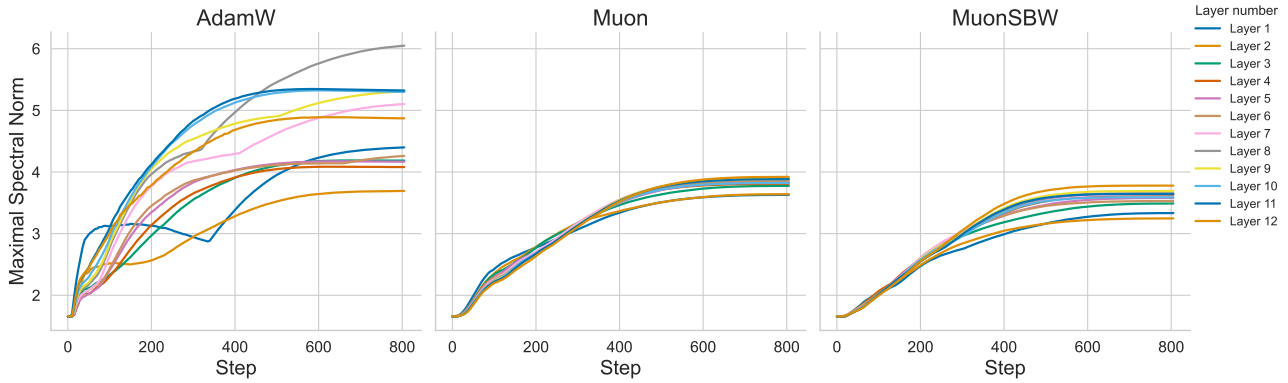

Figure 18. **MuonSBW obtains lower spectral norms than Muon.** For the 124M model, we compute the maximal spectral norms across weight matrices in one layer (block of the transformer) varying the number of row-wise splits.

Lastly, in Figure 18, we can notice a consistent decrease in the spectral norm for all layers, when using MuonSBW, compared to Muon, which in turn attains lower spectral norms compared to AdamW.

F. Critical Batch Size

For a better understanding of MuonS and MuonSBW, we analyze its critical batch size introduced in (Zhang et al., 2025) on the OpenWebText dataset and compare it to AdamW. For this, we use 1x Chinchilla scaling and achieve a loss of 3.2 with a baseline optimizer, AdamW, with the smallest batch size of 2^7 to ensure that we can also achieve it in other settings. We choose the 345M model as this is the smallest model, which does not have big differences in the best validation loss obtained when comparing AdamW, MuonS and MuonSBW (see Figure 2). In addition, for each batch size, we vary 5 learning rates and choose the best. The maximum number of steps for each batch size is such that it preserves the 1x Chinchilla scaling. In this setting, in Figure 19, we observe that CBS for AdamW is higher; however, it requires more steps for each batch size and scales much worse for higher batch sizes. Moreover, it does not achieve the target loss for the largest batch size of 2^{12}.

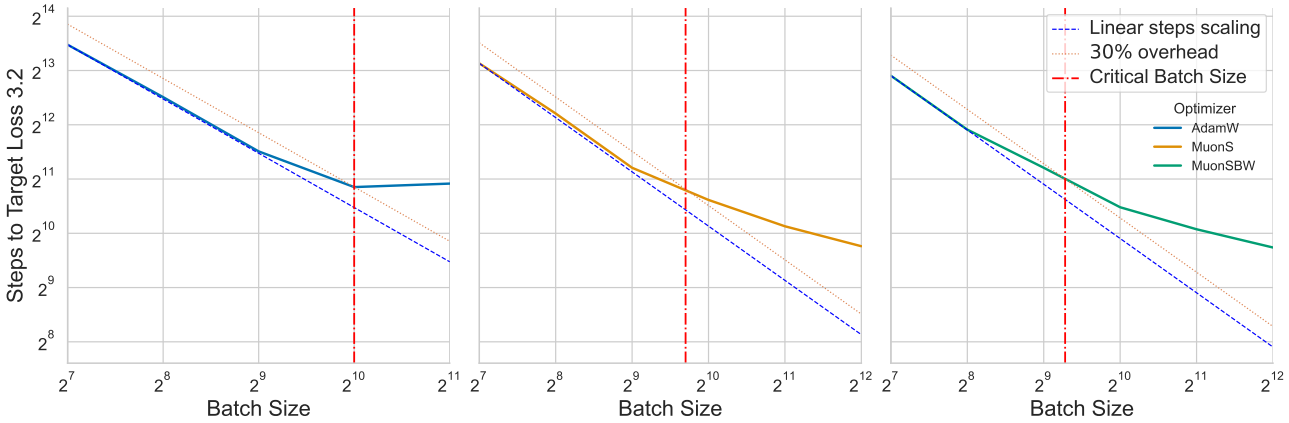

Figure 19. **While CBS for AdamW is higher, it requires more steps for each batch size and scales much worse for higher batch sizes.** For 345M nanoGPT model, we vary batch size during the training using 1x Chinchilla scaling (thus the number of optimization steps is changed accordingly) to understand, how optimizers influence CBS. Note that for batch size 2^{12} AdamW does not reach the target loss.

G. Results for C4 Dataset

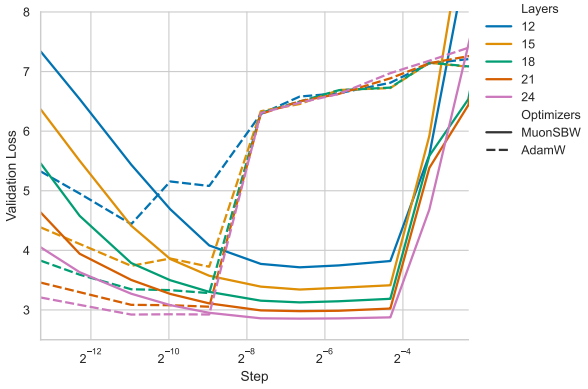

Figure 20. **MuonSBW enjoys learning rate transfer during depth-width-token co-scaling for C4 dataset.** Similar to observations in Figure 2 for OpenWebText, we observe for another dataset, C4, that the MuonSBW learning rate transfers when simultaneously scaling model depth, width, and number of tokens, while for AdamW it does not.

In this section, we investigate how some of the properties of MuonSBW observed with nanoGPT on the OpenWebText dataset transfer to the C4 dataset. We already saw in Figure 1 that MuonSBW has a similar scaling behavior when increasing the number of tensor parallel splits (TP) for OpenWebText and C4. Here, we further compare its learning rate transfer and weight decay influence in the following sections.

G.1. Learning rate transfer

First, we compare the learning rate transfer of MuonSBW and AdamW. We can see in Figure 20 a behavior similar to that we already observed for OpenWebText in Figure 2 – there is learning rate transfer for MuonSBW during depth-width-token co-scaling, unlike for AdamW. Furthermore, the optimal learning rate for OpenWebText of 0.01 is also the best here. Due to the time and compute constraints we train the models here up to 24 layers, while for the experiment in Figure 20 we trained one more size of the model, with 30 layers.

G.2. Static Weight Decay

In addition, for two model sizes, 124M and 345M, in Figure 22 we show the behavior of static weight decay when training with MuonSBW and 5x Chinchilla scaling. Similarly to models trained on OpenWebText (see Section 5.4), we observe that a higher weight decay value of 0.1 outperforms other weight decay values only at the end of the training run.

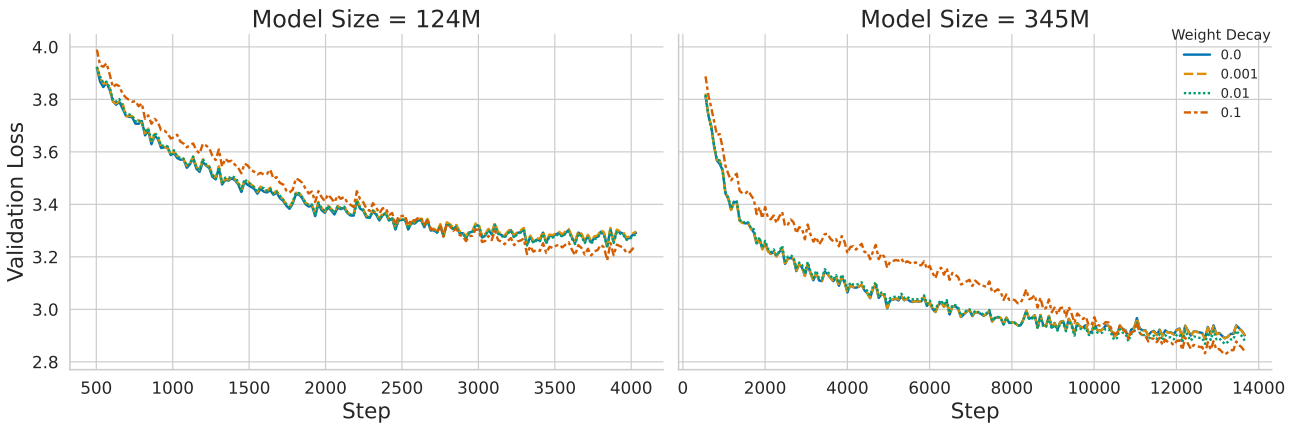

Figure 21. **Higher weight decay of** 0.1 **consistently outperforms only at the end of the training.** Similar to previous observations in Section 5.4, we observe for MuonSBW additionally on C4 dataset, that a higher weight decay value of 0.1 initially performs worse and then, at the end of the training, better than lower constant values in validation loss.

G.3. Dynamic Weight Decay

Finally, we try the weight decay schedules proposed in Section 5.6 for MuonSBW trained on C4. We see that similarly to OpenWebText (see Appendix D.1), increasing Cutoff proportion and the degree in Inverse Polynomial schedule decreases the validation loss. However, it remains comparable to the baseline with the constant weight decay value of 0.1, while on OpenWebText we observe improvement in the validation loss for these both schedules.

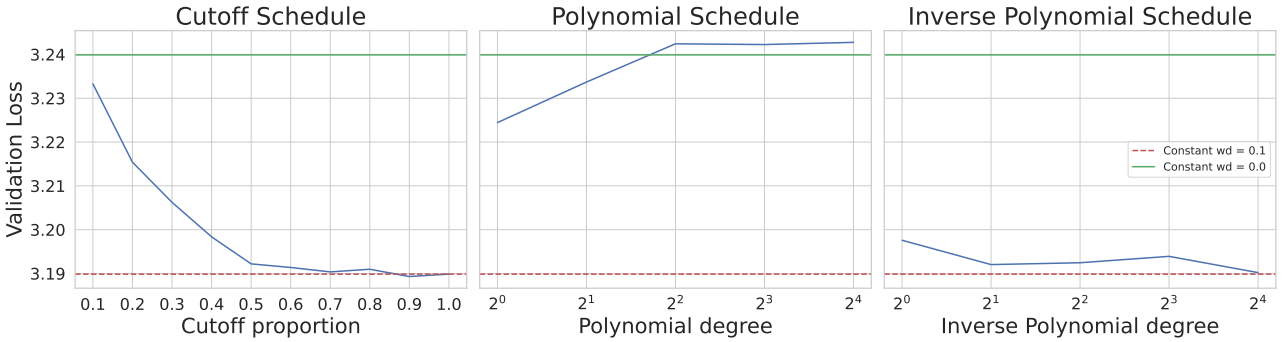

Figure 22. **Clipping weight decay at** 80% **of the training and Inverse Polynomial Schedule.** Motivated by previous observations in Section 5.4, we investigate for MuonSBW additionally on C4 dataset, if clipping weight decay or varying it throughout the training has influence on the performance of the model. We see that clipping it at 90% of the training outperforms the baseline weight decay slightly.

H. Experimental Details

Here, we provide more details of our experimental setting used throughout the paper. They remain consistent across all experiments in the paper, unless we specify otherwise.

H.1. Model Hyperparameters

We used the original nanoGPT model (Karpathy, 2022) without changing its initialization with a block size of 1024 and vocabulary size of 50304 (GPT-2 vocabulary size of 50257, padded up to the nearest multiple of 64 for efficiency). When we increase the number of layers, we consider the following model sizes, together with the number of layers in brackets: 124M (12), 215M (15), 345M (18), 524M (21), 758M (24), 1.43B (30). We do depth-width-token co-scaling, by setting the number of heads in the nanoGPT to be the same as the number of layers, and additionally setting the embedding dimension to be four times the number of heads (and thus layers).

H.2. Optimizer Hyperparameters

For all optimizers, we use the cosine learning rate schedule with the linear warm-up until 2% of the training and decay it until the end of the training. By default, we use 1x Chinchilla scaling, that is, the number of tokens used is twenty times the number of model parameters. Following the nanoGPT codebase, we also use gradient clipping of the global norm at 1.0.

AdamW. By default, we set the weight decay to 0.1 and β_1 with β_2 to 0.9 and 0.95, respectively.

Muon. Spectral norm constraint is used for all layers, but the 1D tensors, together with the first and last layers, are optimized with AdamW. By default, we set the Nesterov momentum to 0.9, AdamW β_1 and β_2 to 0.9 and 0.95, and AdamW weight decay to 0.01. The orthogonalization is approximated with the quintic NS iteration using 6 steps. If not explicitly specified, we use the same learning rate for the layers optimized with the spectral norm constraint and AdamW.

MuonS. We use the same setting as for Muon, with the difference that we use the spectral norm constraint for all layers, and we use AdamW for 1D tensors, unless specified otherwise.

MuonSBW. We use the same setting as for MuonS, however, we perform NS iteration on either row-, column-, or block-wise splits as described in Section 4 and concatenate them afterwards.

Scion. We use the same setting as for MuonS; however, for the first and last layers, we use ℓ_∞ norm constraint, which implies sign updates. Unless otherwise specified, we increase the learning rate for the first and last layers with the ℓ_∞ norm constraint by a factor 10.

H.3. Details About Datasets

OpenWebText (Gokaslan et al., 2019). Train split contains 9B tokens and validation split – 4M tokens.

C4 (Raffel et al., 2019; for AI, 2019). We use the "en" part of the dataset. The train split contains 175B tokens, and the validation split – 87M tokens.

H.4. Details About the Compute

For all our experiments, we were training models using three types of nodes with 8 NVIDIA GPUs each: A100, L40S, and A10G. Each training run was done on one full node, depending on the RAM required.