How to Scale Second-Order Optimization

Zixi (Charlie) Chen* NYU
Shikai Qiu* NYU
Hoang Phan NYU
Qi Lei NYU
Andrew Gordon Wilson NYU

Abstract

Several recently introduced deep learning optimizers inspired by second-order methods have shown promising speedups relative to the current dominant optimizer AdamW, particularly in relatively small-scale experiments. However, efforts to validate and replicate their successes have reported mixed results, with some finding quickly diminishing advantage over AdamW with scale. In this work, we investigate *how to scale* second-order optimizers to achieve optimal performance at scale. Through theoretical and empirical analysis, we derive scaling rules for hyperparameters such as learning rate and weight decay as we scale up model width and depth for a wide range of optimizers, including Shampoo, SOAP, and Muon, accounting for the impact of commonly used techniques such as blocking and grafting. For compute-optimal scaling, we find scaling independent weight decay as $1/\mathrm{width}$ is nearly optimal across optimizers, and that second-order optimizers have a substantially larger optimal model size compared to AdamW for a fixed compute budget. Applying these scaling rules, we show Muon consistently achieves close to $1.4\times$ speedup over AdamW for training transformer models ranging from 190M to 1.4B, while incorrect scaling rapidly diminishes the speedup to around $1.1\times$.

1 Introduction

The non-convex and ill-conditioned loss landscapes of neural networks seem ideally suited for advanced higher-order optimization algorithms. Yet, simple first-order optimizers — AdamW, in particular [11] — continue to dominate the majority of deep learning workloads, including the training of state-of-the-art language models [30, 12]. Recently, however, second-order approaches have regained attention, catalyzed by Shampoo's first place in the AlgoPerf benchmark competitions [22, 33]. Since then, the deep learning community introduced a new wave of optimizers that approximate or take inspiration from second-order methods [35, 20, 23, 5]. The recent success of Muon in training trillion-parameter language models [34] points to the potential in replacing AdamW as the default choice for deep learning at scale.

To understand and fully realize the practical value of second-order optimizers, it is essential to understand the interventions required to effectively employ second-order optimizers as we scale up the models. A key lesson from scaling up deep learning over recent years is that choices such as model size, training horizon, initialization, and learning rate needs to be carefully covaried as the compute budget is scaled to achieve optimal performance [18, 21, 40, 14, 26]. How to systematically generalize existing scaling rules developed for traditional optimizers such as Adam to second-order optimizers is non-trivial, due to their more complex update rules. In practice, practitioners typically rely on heuristics, such as matching the average update entry size to Adam [24, 2, 33] for learning rate scaling, and reusing the 20 tokens per parameter rule for choosing the optimal model size [18], which we will show is suboptimal. We hypothesize the lack of good scaling prescriptions for these recently introduced optimizers may contribute to the inconsistent observations in the literature regarding their relative performance to AdamW [24, 34, 37, 31].

*Equal contribution. Correspondence to zc2157@nyu.edu, sq2129@nyu.edu, andrewgw@cims.nyu.edu.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

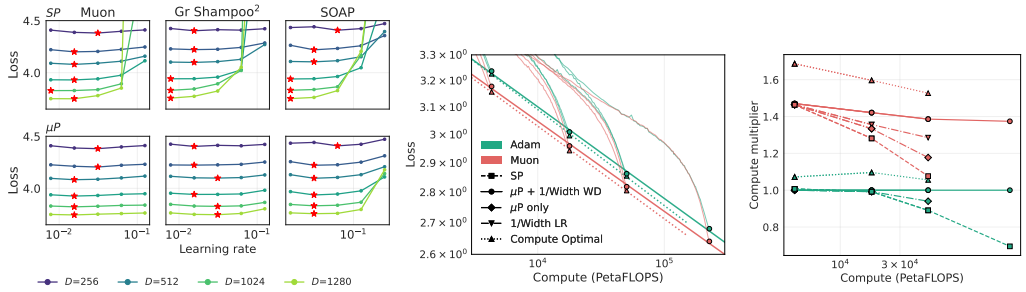

Figure 1: **Proper scaling rules empowers the full potential of second-order optimizers at scale.** (**left**) With our μP learning rate scaling, optimal learning rate is nearly invariant across width D under the 20 token-per-parameter compute-optimal scaling. (**Middle**) Muon consistently outperforms AdamW under our scaling rules that combine μP with $1/D$-scaled weight decay. Ablating either degrades Muon performance significantly. Muon performance further improves by reducing the token per parameter (TPP) from 20 to 7. (**Right**) Estimated compute saving relative to AdamW. Ablating the scaling rules reduces the saving from $1.4\times$ for the 190M model to around $1.1\times$ for the 640M model, highlighting the importance of proper scaling for scaling law comparison.

In this work, we address these open questions by systematically studying scaling rules for commonly used second-order optimizers. We summarize our main findings as follows:

1. We study how to parameterize models trained with second-order optimizers as a function of both width and depth. With a generically applicable technique, we show how to derive the Maximal Update Parameterization (μP) [40, 41] for a wide range of optimizers, including Shampoo [16], SOAP [35], and Muon [20], accounting for the impact of commonly used techniques such as blocking and grafting [2, 33], and extend to depth scaling for residual networks such as the transformer.
2. For fixed training steps, we show the optimal learning rate is stable as predicted by the theory, but with exceptions that reveal the limited applicability of μP in reasoning about the finite-width behavior of models trained with some preconditioners that inflate the stable rank of the update. As an example, we find learning rate transfers better for SOAP if we assume no alignment between the update and the input.
3. For compute-optimal scaling, we find 1) our scaling rules produce nearly stable optimal learning rate, and outperform the standard parameterization with unscaled learning rate, 2) scaling the independent weight decay as $1/\text{width}$ as suggested in Xiao [39] is near-optimal *across optimizers*, and 3) second-order optimizers have larger optimal model size for a fixed compute compared to Adam.
4. We show that applying these scaling rules is crucial for good performance at scale. We find Muon achieves a consistent $1.4\times$ speedup over AdamW in training transformer language models, while incorrect scaling can decrease the speedup from $1.4\times$ to below $1.1\times$ from 190M to 1.4B parameter models.

We make our code available here.

2 Background

Second-Order Optimizers in Deep Learning. Natural Gradient Descent (NGD), a classical second-order optimizer, accelerates convergence by preconditioning the gradient with the inverse of the empirical Fisher information matrix. AdaGrad [13] maintains the cumulative outer product of gradients $\Omega_t = \sum_{s=1}^{t} g_s g_s^T$ and preconditions updates with $\Omega_t^{-1/2}$. For cross-entropy loss, AdaGrad approximates NGD, up to step-size difference. However, computing a full Fisher is infeasible even for a moderately-sized model, necessitating structured approximation in practice. Adam [11] adopts a diagonal approximation, while K-FAC [15] exploits a Kronecker-factored representation.

Shampoo [16, 2, 33] is an effective second-order optimizer that demonstrated strong performance in a comprehensive optimization benchmark [22]. Its preconditioner can be interpreted as a one-step

approximation of the empirical Fisher using power iteration [27]. Let W_t denote the weight matrix of a given layer and G_t the gradient estimate at W_t at time t. The Shampoo update rule is given by

$$W_{t+1} = W_t - \eta(L_t + \epsilon I)^{-e_L} G_t (R_t + \epsilon I)^{-e_R}, \tag{1}$$

where η is the learning rate, ϵ is a regularization parameter, and e_L and e_R are tunable positive exponents. In this work, we refer to the update rule with $e_L = e_R = 1/4$ as Shampoo [16] and $e_L = e_R = 1/2$ as Shampoo^2 [27, 33]. $L_t \in \mathbb{R}^{m\times m}$ and $R_t \in \mathbb{R}^{n\times n}$ are given by $L_t = \mathbb{E}[G_t G_t^T]$ and $R_t = \mathbb{E}[G_t^T G_t]$, with the expectation estimated using an exponential moving average over the trajectory.

A key weakness of Shampoo is that the matrix inversion scales cubically with the width of layer and is numerically unstable. To make Shampoo practical, *blocking* [33, 35] partitions the parameter into sub-blocks, thereby decreasing the size of the preconditioners; *infrequently updating preconditioners* [33, 35] amortizes the cost of matrix inversion. *Grafting* [2, 33, 1], which is used to enhance stability, renormalizes the Shampoo update by another first-order method, usually Adam.

Following Shampoo's strong performance in the AlgoPerf benchmark [22], several new second-order optimizers have emerged. Bernstein and Newhouse [5] showed that without exponentially moving average, Shampoo simplifies to:

$$W_{t+1} - W_t = -\eta \cdot (G_t G_t^T)^{-1/4} G_t (G_t^T G_t)^{-1/4} = -\eta \cdot U_t V_t^T, \tag{2}$$

where $G_t = U_t \Sigma_t V_t^T$ is the SVD of the gradient matrix. The Muon optimizer [20] leverages this insight, employing an iterative method to compute $U_t V_t^T$ directly. Muon has gained traction due to its algorithmic simplicity and robust performance. While the community disagrees on whether Muon should be considered second-order, we include in our analysis due to its promising performance and close connection to methods like Shampoo.

SOAP [35], another Shampoo-inspired algorithm, recognizes that Shampoo effectively applies AdaFactor in a transformed coordinate system before mapping back to the original parameter space. SOAP extends this insight by applying Adam-style updates within this transformed coordinate representation. By a slight abuse of notation, the update is given by:

$$W_{t+1} = W_t - \eta Q_L \,\mathrm{Adam}(Q_L^T G_t Q_R) Q_R^T, \tag{3}$$

where $\mathrm{Adam}(\cdot)$ is the Adam update rule. The orthogonal matrices $Q_L \in \mathbb{R}^{m\times m}$ and $Q_R \in \mathbb{R}^{n\times n}$ are given by $Q_L = \mathrm{eigvec}(\mathbb{E}[G_t G_t^T])$ and $Q_R = \mathrm{eigvec}(\mathbb{E}[G_t^T G_t])$, where $\mathrm{eigvec}(A)$ gives the eigenvectors of A. We provide a more detailed version of update rules in Appendix A.

Feature Learning and Hyperparameter Transfer. The *maximal-update parametrization* (μP) [40] formalizes how learning rate should be scaled with width to maintain non-negligible feature update in the infinite width limit by ensuring the amount by which the features change per step is consistent across widths. Under μP, hyperparameters (HPs) tuned on a small model can be transferred to models that are orders of magnitude wider with little or no re-tuning, dramatically reducing training cost [43]. To extend feature learning to a joint infinite-width–infinite-depth limit, subsequent works have proposed to multiply the residual branch by $1/L^\alpha$ and scale the learning rate accordingly to ensure the rate of feature learning is consistent across depth, where L is the depth and $\alpha \in [0.5, 1]$ [45, 6, 8, 10]. Together, principled width and depth scalings provide learning rate transfer for scaling first-order optimizers like SGD and Adam, but analogous prescriptions for second-order methods remain largely unexplored, with the exception of the recent work by Ishikawa and Karakida [19] deriving μP for Shampoo and K-FAC. Throughout our experiments, we use a number of variants of Shampoo and Shampoo^2, such as blocking grafting by Adam, which require different scalings compared to what is derived in Ishikawa and Karakida [19] and have been shown to perform better [33].

Compute-Optimal Scaling. Empirical scaling laws show that test loss falls as a power law in model parameters and data[21, 18]. Yet existing analyses assume first-order updates; how curvature-aware optimizers reshape these laws, or alter constants like the Chinchilla token-per-parameter ratio, has been an open question. We show that second-order optimizers indeed alter compute-optimal scaling laws: they are often significantly more sample-efficient and require much less data to be computate-optimal.

3 Hyperparameter Transfer Over Width and Depth

In this section, we present our scaling rules for the per-layer learning rate and regularization parameter ϵ as a function of width and depth to ensure consistent feature learning as the model scales, summa-

Table 1: Summary of hyperparameter scaling rules for learning rate η and damping parameter ϵ as a function of the input dimension d_{in}, output dimension d_{out}, depth L, and number of blocks N_{blk} as used in blocking. $A = \frac{d_{\text{out}}}{d_{\text{in}}}$ is the aspect ratio. A multiplier $1/L$ is assumed to apply to the output of every residual block. For parameters outside of the residual blocks (e.g. embedding and last layer), set $L = 1$. η_{Q_2} denotes the correctly scaled learning rate under preconditioner Q_2. We include SGD and Adam here for completeness. The exact definition of ϵ is described in Appendix A.

	SGD	**Adam**	**Shampoo** (e_L, e_R)	**SOAP**	**Muon**	**Grafting** $Q_1 \to Q_2$
η	LA	$\frac{1}{d_{\text{in}}}$	$L^{1-2(e_L+e_R)}A^{1-e_L-e_R}/N_{\text{blk}}^{e_L+e_R}$	$\sqrt{A}$	$\sqrt{A}$	η_{Q_2}
ϵ	N/A	$\frac{1}{Ld_{\text{out}}}$	$\frac{1}{L^2AN_{\text{blk}}}$	$\frac{1}{L\sqrt{A}}$	$\eta_Q^{-1}\sqrt{A}$	$\eta_{Q_1}^{-1}\sqrt{A}$

rized in Table 1. We provide a high-level overview of the derivation here and leave the full details to Appendix B. We adopt Big-Theta notation along with an elementwise convention for vectors. Specifically, for vector $v \in \mathbb{R}^d$, $v = \Theta(g(d))$ is a shorthand for $\|v\|_{\text{RMS}} = \Theta(g(d))$, where the normalized root mean square is defined by $\|v\|_{\text{RMS}} \equiv \sqrt{\frac{1}{d}\sum_i v_i^2}$. We use the notation $a \sim b$ to indicate $a/b = \Theta(1)$. We provide the experiment details in Appendix D.

3.1 Width Scaling via μP

In this section, we focus on how hyperparameters should be scaled as a function of width for a wide variety of optimizers by analyzing their Maximal Update Parameterization (μP) [40, 42, 41, 44]. μP ensures the rate of feature learning is consistent across width in every layer and that the training dynamics converges to a well-defined large-width limit.

We now show a simple and general procedure suffices for determining the μP scaling applicable to a wide range of second-order optimizer used in deep learning, each requiring only a few lines of derivation. For concreteness, we consider training an L-layer (L assumed fixed for now) MLP that outputs $f(\xi)$ on an input ξ as follows:

$$x_0(\xi) = \xi,\ h_\ell(\xi) = W_\ell x_{\ell-1}(\xi),\ x_\ell(\xi) = \phi(h_\ell(\xi)),\ \ell = 1, \ldots, L-1,\ f(\xi) = h_L(\xi) = w_L^\top x_{L-1}(\xi),$$

where weights are drawn from a Gaussian $\mathcal{N}(0, \sigma_\ell^2)$ with layerwise variances σ_ℓ^2 and ϕ is an elementwise nonlinear function. For simplicity, we assume $\xi \in \mathbb{R}, f(\xi) \in \mathbb{R}$, $h_\ell(\xi) \in \mathbb{R}^d$ for $\ell = 1, \ldots, L-1$, and refer to d as the width.

We will now analyze the conditions for μP up to the first gradient step. By satisfying these conditions, the network will be in μP in all subsequent steps by induction. To do so, we track the change in layer outputs on a generic input ξ' induced by the first gradient update on ξ. We denote the change in any quantity X after this step as ΔX. Extension to batch size B greater than 1 is straightforward as long as B is constant in width, which we discuss in Appendix C.

Initialization and Gradient Scales are Optimizer-Independent. Before discussing specialization to any particular optimizer, we make the observation that the initialization scale σ_ℓ of W_ℓ is independent of the optimizer, with $\sigma_\ell = \Theta(1/\sqrt{d_{\text{in},\ell}})$ for $\ell < L$ and $\sigma_L = \Theta(1/d)$. This result follows from the combination of stability and feature learning condition of μP. Specifically, to ensure stability at initialization, μP requires all activations in the hidden layers have $\Theta(1)$ entries and that the function output is $O(1)$:

$$h_\ell(\xi') = W_\ell x_{\ell-1}(\xi') = \Theta(1),\ \ell = 1, \ldots, L-1, \quad f(\xi') = O(1) \tag{4}$$

The first condition implies that W_ℓ have entries of scale $\sigma_\ell = \Theta(1/\sqrt{d_{\text{in},\ell}})$ for all but the last layer, where $d_{\text{in},\ell}$ is the input dimension of the ℓ layer ($d_{\text{in},\ell} = 1$ if $\ell > 1$ and 1 if $\ell = 1$), while second condition implies that $\sigma_L = O(1/\sqrt{d})$. To then ensure non-negligible feature learning, i.e. the change in the last layer feature $\Delta x_{L-1}(\xi')$ is $\Theta(1)$ per step, while ensuring the resulting change in the output $\Delta f(\xi')$ is $\Theta(1)$ (predictions are updated without diverging), the last layer weights w_L must be $\Theta(1/d)$. To see that, note $\Delta f(\xi') = w_L^\top \Delta x_{L-1}$ is a sum of d i.i.d. random variables, each with a generically non-zero mean due to the correlation between elements in w_L and elements in Δx_{L-1}. The correlation is induced by backpropagation, causing Δx_{L-1} to be a function of w_L, regardless of the specific optimizer. Therefore, given Δx_{L-1} is $\Theta(1)$, w_L and thus σ_L must be $\Theta(1/d)$.

As a result, the entries of the gradient of the loss w.r.t. the pre-activations $\delta_\ell(\xi') \equiv \nabla_{h_\ell(\xi')} L$ scales as $\Theta(1/d_{\text{out},\ell})$ since backpropagating through all hidden layers preserve the scale of the gradient given that $\sigma_\ell = \Theta(1/\sqrt{d_{\text{in},\ell}}) = \Theta(1/\sqrt{d_{\text{out},\ell}})$ for $1 < \ell < L$. With these results, we are now ready to state the simplified μP condition for any optimizer.

Maximal Update For Any Optimizer. To ensure each layer is subsequently updated as much as possible and not divergent, μP requires each layer produces $\Theta(1)$ update to its output for every gradient step. Let ΔW_ℓ denote the update to W_ℓ due to training on $\xi \in \mathbb{R}$, this condition states

$$\Delta W_\ell x_{\ell-1}(\xi') = \Theta(1),\ \ell = 1, \ldots, L. \tag{5}$$

The gradient of W_ℓ computed on a datapoint $\xi \in \mathbb{R}$ is $G_\ell(\xi) \equiv \nabla_{W_\ell} L(\xi) = \delta_\ell(\xi) x_{\ell-1}(\xi)^\top$, where $\delta_\ell(\xi) \in \mathbb{R}^{d_{\text{out},\ell}}, x_\ell(\xi) \in \mathbb{R}^{d_{\text{in},\ell}}$. Let $Q(G_\ell(\xi))$ be the preconditioned gradient returned by the optimizer, we therefore wish to choose learning rate η_ℓ such that $\eta_\ell Q(G_\ell(\xi)) x_{\ell-1}(\xi') = \Theta(1)$, where

$$G_\ell(\xi) = \delta_\ell(\xi) x_{\ell-1}(\xi)^\top, \delta_\ell(\xi) = \Theta(1/d_{\text{out},\ell}), x_{\ell-1}(\xi)^\top x_{\ell-1}(\xi') = \Theta(d_{\text{in},\ell}). \tag{6}$$

Moving forward, we will omit the layer subscripts and abbreviate $x(\xi)$ as x, $x(\xi')$ as x', $\delta(\xi)$ as δ, and $G(\xi)$ as G. As a result, the following holds for all layers:

$$x = \Theta(1), \quad x' = \Theta(1), \quad x^\top x' = \Theta(d_{\text{in}}), \quad \delta = \Theta(1/d_{\text{out}}), \quad \delta^\top \delta = \Theta(1/d_{\text{out}}). \tag{7}$$

It turns out these scaling relations are sufficient to determine the μP scaling rules. With these relations, solving for the learning rate (and additional hyperparameters) now reduces to straightforward algebraic manipulations. To illustrate, we now show how to derive the scaling rules for Shampoo. We provide full derivations for all optimizers in Table 1 in Appendix B. Our result for Shampoo reproduces that in Ishikawa and Karakida [19].

Example: Shampoo. Let $e_L, e_R \in [0, 1]$. The rank–1 Gram matrices are $L = \delta\,(x^\top x)\,\delta^\top, \qquad R = x\,(\delta^\top \delta)\,x^\top$ with unique nonzero eigenvalues $\lambda_L = \lambda_R = (x^\top x)(\delta^\top \delta) = \Theta\Big(\frac{d_{\text{in}}}{d_{\text{out}}}\Big)$. The preconditioned update is

$$Q_{\text{Shampoo}}(G) = (L + \epsilon_B I)^{-e_L}\, G\, (R + \epsilon_A I)^{-e_R}. \tag{8}$$

Since $G = \delta x^\top$ lies in the principal directions,

$$Q_{\text{Shampoo}}(G)\, x' \sim (\lambda_L + \epsilon_B)^{-e_L} (\lambda_R + \epsilon_A)^{-e_R}\, \delta\, (x^\top x') = \Theta\Bigg(\Big(\frac{d_{\text{in}}}{d_{\text{out}}}\Big)^{1-e_L-e_R}\Bigg), \tag{9}$$

hence $\eta_{\text{Shampoo}} = \Theta\Bigg(\Big(\frac{d_{\text{out}}}{d_{\text{in}}}\Big)^{1-e_L-e_R}\Bigg)$. To keep the action of the regularization on the active directions neither trivial nor ill-conditioned, ϵ_A, ϵ_B must balance the only large scale there, namely $\lambda_{L,R} = \Theta\Big(\frac{d_{\text{in}}}{d_{\text{out}}}\Big)$, so $\epsilon_A,\ \epsilon_B = \Theta\Big(\frac{d_{\text{in}}}{d_{\text{out}}}\Big)$.

Example: SOAP. SOAP applies Adam in the eigenbasis U and V of the Shampoo preconditioners L and R. The top eigenvectors are $u_1 = \delta/\|\delta\|$ and $v_1 = x/\|x\|$. Then

$$G' = U^\top G V = (\|\delta\|\,\|x\|)\, e_1 e_1^\top, \qquad \|\delta\|\,\|x\| = \Theta\Bigg(\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\Bigg). \tag{10}$$

Ignoring momentum, SOAP applies elementwise $\widehat{G}' = (|G'| + \epsilon)^{-1} \odot G'$. To avoid degeneracy, ϵ must match the active magnitude, so $\epsilon_{\text{SOAP}} = \Theta\Bigg(\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\Bigg)$. Transforming back,

$$Q_{\text{SOAP}}(G) = U\, \widehat{G}'\, V^\top = \Theta(1) u_1 v_1^\top, \qquad Q_{\text{SOAP}}(G)\, x' = \Theta(1)\, \delta\, \frac{x^\top x'}{\|\delta\|\,\|x\|} = \Theta\Bigg(\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\Bigg), \tag{11}$$

which yields $\eta_{\text{SOAP}} = \Theta\Bigg(\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}\Bigg)$.

Example: Grafting. Grafting normalizes Shampoo update's Frobenius norm by that of Adam's. As can be seen from the derivation for Shampoo and SOAP, the main insight is that in the infinite-width limit, the gradient and the preconditioners are low-rank, with *aligned* singular vectors. As a result, the Shampoo update remains (stable) low rank, implying that its Frobenius norm scales identically with its spectral norm. Prior works have shown that Adam updates also have $\Theta(1)$ stable rank [44, 41]. Consequently, grafting by Adam requires simply scaling the learning rate with the μP scaling for Adam.

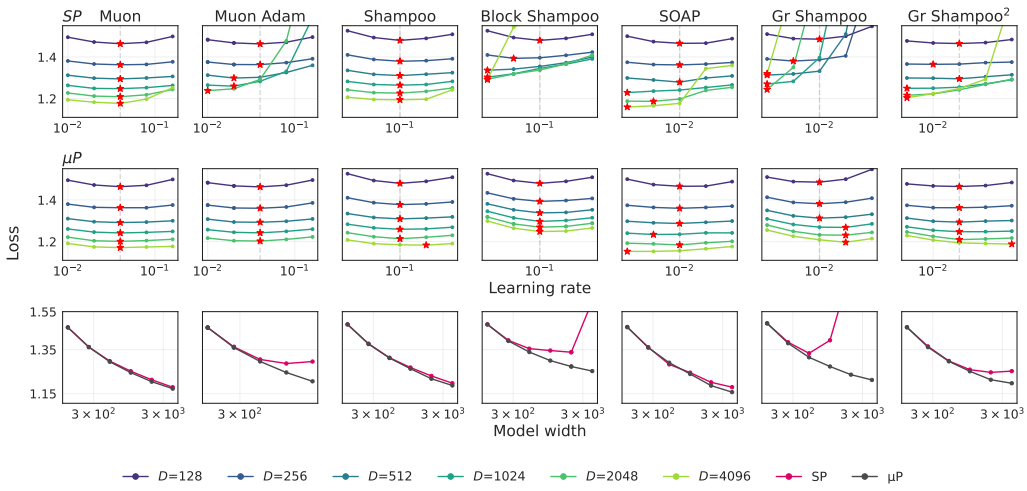

Figure 2: μ**P for scaling learning rate with width in second-order optimizers on OpenWebText.** (**Top row**) We show that the optimal learning rate shifts in the standard parameterization (SP) for seven variants of second-order optimizers. (**Middle row**) Our proposed scaling rules, on the other hand, keep the optimal learning rate stable across widths. (**Bottom row**) μP enables better zero-shot transferring of the optimal learning rate found in the smallest model to larger models. The learned features for both scaling approach are provided in Appendix D.2 to further justify the correctness of our approach. Muon Adam uses Adam for embedding and readout layers. Block Shampoo adopts blocking with block size 128. 'Gr' stands for grafting by Adam.

3.2 Empirical Validation of μP and Diagnosis of Poor Transfer

In Figure 2, we validate our μP scaling rules on transformers trained on the OpenWebText dataset for 100M tokens. We use a small vocabulary with only 96 unique tokens so we can feasibly apply the full preconditioners on the embedding and readout layers. We describe the full experiment setup in Appendix D. Across seven optimizers, we find μP led to significantly more stable optimal learning rate as width scales from $D = 128$ to $D = 4096$, and consistently outperform SP when zero-shot transferring the optimal learning rate found on the $D = 128$ model (indicated by the dashed vertical line).

Grafting by Frobenius Norm Breaks Learning Rate Transfer. Upon further inspection, we see that the optimal learning rate still drifts noticeably in μP for Grafted Shampoo, Grafted Shampoo^2, and SOAP. In Figure 3(left), we show the increasing optimal learning rate for Grafted Shampoo is due to a transition from the stable rank scaling as $\Theta(1)$ to growing with D. As training progresses, the preconditioner update leaves the asymtotically large D, low-rank regime, causing the Frobenius normalization to undershoot the spectral norm of the update for large D in the hidden layers, resulting in slower feature evolution and increasing optimal learning rate. While the same transition occurs for the vanilla Shampoo, the absence of Frobenius normalization preserves the right spectral norm and feature update scale. Based on this observation, we argue gratfting should be done in the spectral norm rather than the Frobenius norm, in order to enable learning rate transfer for preconditioners that signifcaintly inflate the stable rank. The exception is if we use blocking with a $\Theta(1)$ block size, which caps the stable rank.

Corrected SOAP Scaling by Assuming No Alignment. In Figure 3(right) SOAP, we observe the opposite trend for SOAP, where feature updates grow with D as training progresses, leading to a decreasing optimal learning rate. We hypothesize this is due to SOAP amplifying entries in G' (Equation (10)) which vanish in the infinite-width limit but remain non-zero at finite width by applying Adam in the eigenbasis, producing a dense rather than a sparse $\widehat{G}'$ that aligns weakly with the input. Thus, we introduce a corrected scaling rule whereby we assume no alignment between the SOAP update and the input, scaling η as $1/\sqrt{d_{\text{in}}}$ instead. Figure 3(d) shows this correction leads to more stable optimal learning rate and consistent feature updates (after an initial transient). We adopt this corrected scaling rule SOAP in the subsequent experiments.

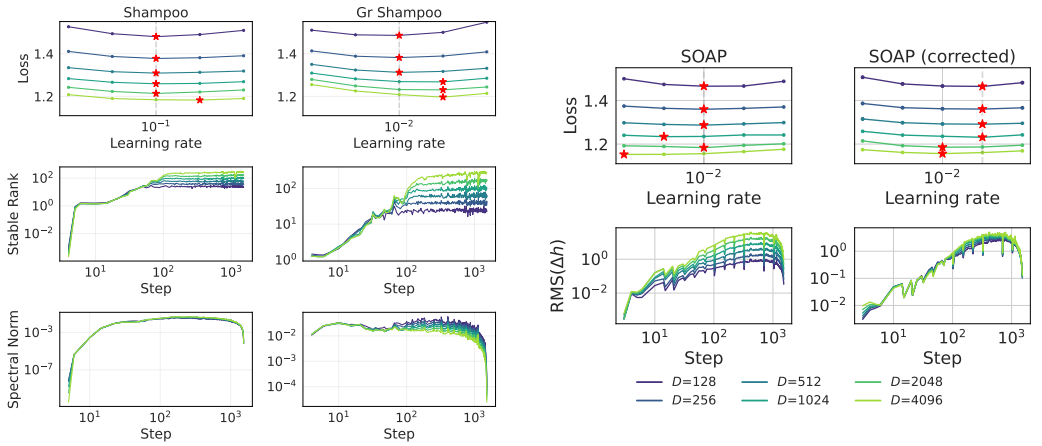

Figure 3: **Diagnosing poor learning rate transfer in μP for Grafted Shampoo and SOAP on OpenWebText**. (**Left**) μP scaling fails to transfer the optimal learning rate for Grafter Shampoo, where the scale of the feature updates across consecutive steps decreases with width as training progresses due to spectral norm shrinking with D as a result of Frobenius normalization and growing stable rank. By contrast, vanilla Shampoo's feature updates are stable despitete growing stable rank. (**Right**) Learning transfer fails for SOAP due to diverging feature update size with D. We identify a different scaling of the learning rate corresponding to assuming no alignment between the update and the input, leading to a more stable optimal learning rate and consistent feature update across widths.

3.3 Depthwise Hyperparameter Transfer

We now consider depth scaling for architectures with residual blocks (e.g. the transformer). In the MLP example, we replace Equation (4) with

$$h_\ell = W_\ell x_{\ell-1},\ x_\ell = x_{\ell-1} + \phi(h_\ell). \tag{12}$$

In general, the residual block can be more complex, such as involving layer normalization and multiple matrix multiplies as in the case of a transformer. Recent work has shown that by scaling down the output of each residual block by $1/L$ while ensuring $\Theta(1)$ feature learning inside each block leads to well-defined and performant depth scaling limits for transformers trained with SGD and Adam [8, 10], referred to as the CompleteP. We derive the depth scaling rules for second-order optimizers based on the same principle. Given our setup so far, the only change to the previous derivation in the width-only scaling case is that gradients in the residual blocks now scale as $\frac{1}{Ld_{\text{out}}}$ rather than $\frac{1}{d_{\text{out}}}$. Propagating these additional L-dependent factors to the solution of η and ϵ scaling leads to the final width-depth joint scaling rules in Table 1. We include detailed derivations in Appendix B.

Figure 4 (left) verifies the depth scaling rules yield more stable optimal learning rate across depths from $L = 3$ to $L = 96$, and consistently lower loss when zero-shot transferring learning rate from the 3 layer model. While we observe less sensitivity to mis-scaled learning rate for depth scaling in SP, we verify the depth scaling rules are necessary for consistent feature learning across depths (Figure 4 right).

4 Compute-Optimal Scaling Rules

Equipped with the tools for scaling hyperparameters across model sizes, we now investigate how to scale second-order optimizers in the compute-optimal regime where training tokens are scaled jointly with model size [18, 21]. In this section, we switch to FineWeb with full vocabulary. We detail the experiment setup in Appendix D.

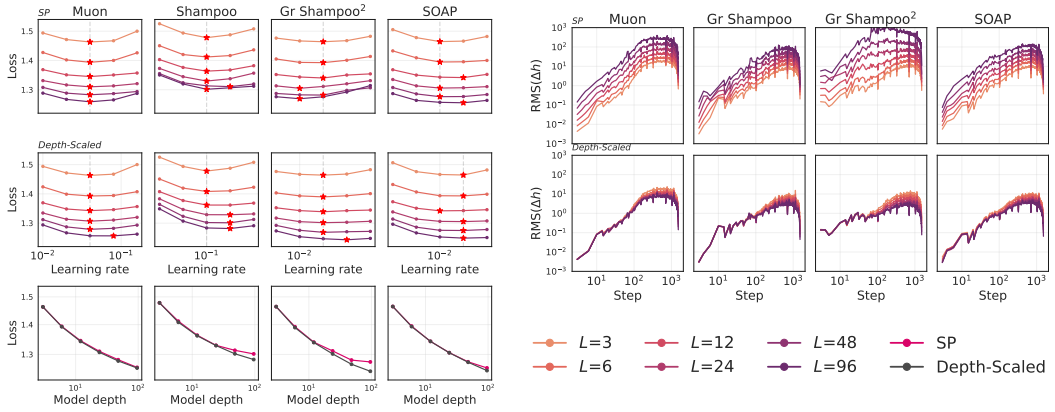

Figure 4: **Depth scaling on OpenWebText**. Our depth scaling rules, which apply a $1/L$ residual branch multiplier and depth-scaling factors to the learning rate, improve the stability of the optimal learning rate (left), and ensure that the feature learning scale is consistent as the depth increases (right). With SP, even though the optimal learning rate looks relatively stable, the RMS of feature updates measured between consecutive evaluation steps shows a clear trend of diverging with depth. Zero-shot transferring the optimal learning rate (the gray dashed line) yields better performance as depth increases with our depth scaling rule (bottom left).

4.1 Approximately Stable Learning Rate in μP

While μP learning rate scaling is derived assuming $\Theta(1)$ training steps, we find it also helps stabilize the optimal learning rate in the 20 tokens per parameter compute-optimal regime (Figure 5 left), in contrast to SP where the optima shift significantly towards smaller values. We apply our scaling rules in the subsequent compute-optimal scaling experiments, but note there are mixed results in the literature regarding whether learning rate scaling derived for $\Theta(1)$ training time translates to compute-optimal scaling [14, 10].

4.2 $1/D$-scaled Weight Decay (WD)

Properly specifying the weight decay is crucial for achieving optimal performance at scale [37, 36, 38]. In Figure 5 (right), we show the combination λD is stable across scale under compute-optimal training, where λ is the independent (decoupled) weight decay [25], corresponding to the $\lambda = \Theta(1/D)$ scaling proposed in Xiao [39]. By default, we adopt this scaling for subsequent experiments. Note this differs from the $\Theta(1)$ weight decay prescribed by μP, as well as the constant weight decay EMA time-scale in [36, 3, 4].

4.3 Optimal Token to Parameter Ratio

One key question relevant to compute-optimal scaling is whether second-order optimizers alter any of the constants and exponents in scaling laws of the form $L(t, P) = (P/P_0)^{-\alpha} + (t/t_0)^{-\beta}$ commonly observed in natural data [17, 32] where P is the number of parameters and t indexes the training tokens. As the first term, referred to as model bottleneck [7], describes loss at convergence, we expect both P_0 and the exponent α to be fixed across optimizers. For any optimizer that outperforms Adam, the second term known as time-bottleneck [7] must improve, by either decreasing the constant t_0 or increasing the exponent β. If β is improved, then the optimal token scales with a smaller exponent with respect to parameters. Otherwise, if only t_0 is improved, then the optimal token per parameter would be reduced by a multiplicative factor.

In either case, we expect optimizers that train faster than Adam to prefer a larger model size at a fixed compute budget. Surprisingly, prior work studying the efficiency of second-order optimizers have neglected to validate or quantify this effect. We verify that Muon indeed requires fewer tokens than Adam to achieve compute-optimal in Appendix D.

5 Scaling Law Comparison

Params	Seq Len	Hidden Dim	Inter Dim	# Layers	# Heads
190M	1024	512	2048	32	8
380M	1024	768	3072	32	12
640M	1024	1024	4096	32	16
1.4B	1024	1536	6144	32	24

Table 2: Architecture specifications for each model size we studied.

In this section, we compare the scaling behavior of Muon and AdamW when training larger Transformer models on the FineWeb dataset. Our setup closely follows Wen et al. [37], with the main difference being the dataset: all models here are trained on FineWeb tokenized using the GPT-2 tokenizer. We use the LLaMA architecture, matching width and depth configurations from Wen et al. [37]. For completeness, the model specifications are provided in Table 2. We adopt a context length of 1024 and batch size of 128. Unless stated otherwise, hyperparameter tuning is performed on the *base* model—32 layers, embedding dimension 512, and 190M parameters. For Muon, the embedding layers use Adam for optimization.

To improve tuning efficiency, we divide hyperparameters into subsets and tune them sequentially via grid sweeps. At each stage of tuning, previously selected hyperparameters remain fixed. After determining optimal settings for the base model, we fix compute budgets and tune model widths to find the optimal tokens-per-parameter ratio using Approach 2 from Hoffmann et al. [18]. Further tuning details are provided in Appendix D.1.

During scaling, we apply the learning-rate and weight-decay scaling rules from Section 4, while keeping all other hyperparameters constant. In particular, we fix the warmup tokens, and we also ablate the alternative strategy that keeps the warmup-token ratio fixed. To assess the robustness of our scaling rule, we perturb the scaled learning rate and weight decay by factors of $2\times$ and $0.5\times$. As shown in Figure 6, these ablations confirm that our scaling rule yields an approximately optimal configuration.

As shown in Figure 7, combining μP learning-rate scaling with $1/D$-scaled weight decay allows Muon to outperform AdamW across model sizes. Muon consistently achieves a $\sim 1.4\times$ speedup on models ranging from 190M to 1.4B parameters using the default 20-TPP setting, and compute-optimal TPP values further improve Muon's efficiency.

To isolate which components of our scaling rule drive this improvement, we perform several ablations, summarized in the right panel of Figure 7. Scaling only the learning rate with μP while keeping weight decay constant places performance between SP and our full approach. Scaling both learning

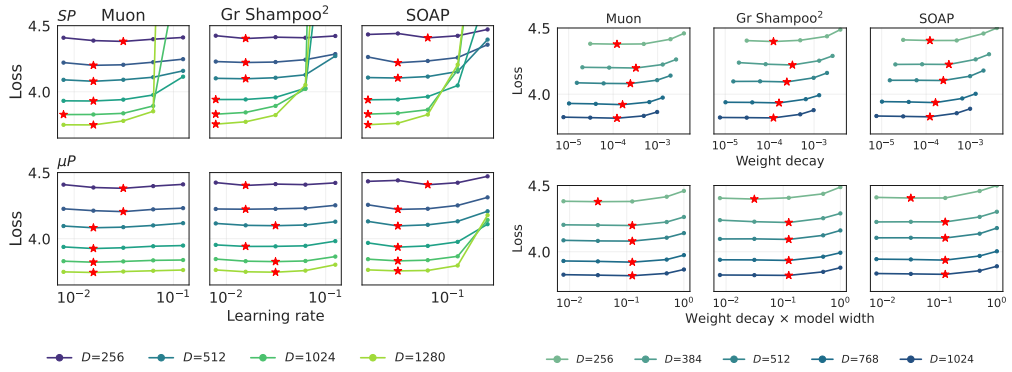

Figure 5: **Stable optimality of learning rate and weight decay under our proposed μP and $1/D$ weight-scaled weight decay on FineWeb**. With μP and $1/D$-scaled weight decay, we observe that the optimal learning rate (**left**) and weight decay (**right**) are roughly consistent across model scales under compute-optimal training with 20 tokens per parameter on FineWeb.

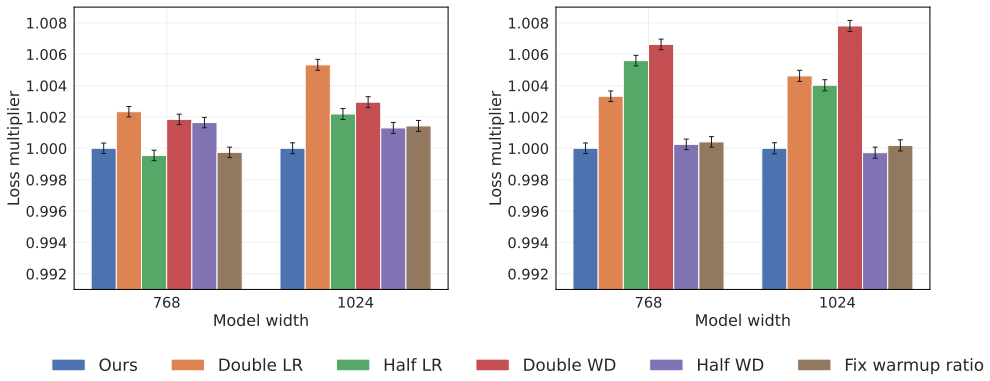

Figure 6: **Loss multiplier under different hyperparameter perturbations.** The loss multiplier is defined as the ratio between the loss of a perturbed configuration and the baseline loss obtained using our scaling strategy. Error bars reflect a ± 0.001 variation in loss, corresponding to the observed randomness across different random seeds. Perturbing the scaled hyperparameters—either doubling or halving the learning rate or weight decay—results in comparable or worse performance, indicating that our scaling approach yields near-optimal hyperparameter settings.

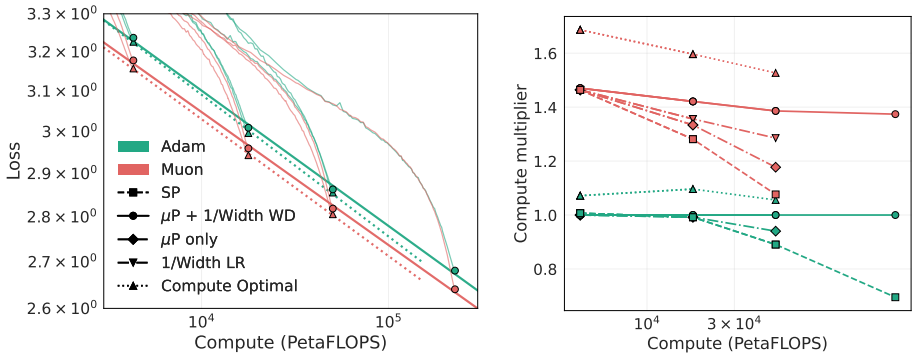

Figure 7: **Muon improve Compute-Optimal Scaling Laws of transformers trained on FineWeb Over AdamW** (Left) Loss vs compute. Muon consistently outperforms AdamW under our scaling rules that combine μP with $1/D$-scaled weight decay. Muon performance further improves by reducing the token per parameter (TPP) from 20 to 7. (Right) Estimated compute multipliers quantify how much additional compute Adam would require to match the loss achieved by the optimizer under the particular scaling configuration. Adam's equivalent compute is estimated by shifting its loss curve using the observed loss gap and the fitted power-law slope in the log-log scale. Removing either the learning-rate scaling or the weight-decay scaling rapidly closes the gap between Muon and Adam as model size increases.

rate and weight decay by $1/\text{width}$—similar to a pure μP rule for Adam—improves over scaling the learning rate alone, but still falls short of our method. These results indicate that both components of our scaling strategy are necessary to achieve the best observed performance.

6 Discussion

Since its introduction over a decade ago, the Adam optimizer has dominated deep learning practice. It was not until very recently that modern second-order approaches such as Shampoo and Muon started to show significant promise in training large-scale neural networks. To fully realize the potential of second-order optimizers, a robust understanding of how to effectively apply them at scale is essential. By studying hyperparameter scaling rules of second-order optimizers, we have shown a number of important scaling considerations and demonstrated proper scaling is essential to consistent efficiency gains across model sizes.

Acknowledgments and Disclosure of Funding

We thank Martin Marek, Andres Potapczynski, and Sanyam Kapoor, for helpful discussions. This work was supported by Google's TPU Research Cloud (TRC) program: https://sites.research.google/trc/.

References

[1] Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Disentangling adaptive gradient methods from learning rates. *arXiv preprint arXiv:2002.11803*, 2020.

[2] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

[3] Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in llm pre-training. *arXiv preprint arXiv:2505.13738*, 2025.

[4] Shane Bergsma, Bin Claire Zhang, Nolan Dey, Shaheer Muhammad, Gurpreet Gosal, and Joel Hestness. Scaling with collapse: Efficient and predictable training of llm families. *arXiv preprint arXiv:2509.25087*, 2025.

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

[6] Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. *arXiv preprint arXiv:2309.16620*, 2023.

[7] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. *arXiv preprint arXiv:2402.01092*, 2024.

[8] Blake Bordelon, Hamza Chaudhry, and Cengiz Pehlevan. Infinite limits of multi-head transformer dynamics. *Advances in Neural Information Processing Systems*, 37:35824–35878, 2024.

[9] Jie Chen and Edmond Chow. A newton-schulz variant for improving the initial convergence in matrix sign computation. *Preprint ANL/MCS-P5059-0114, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL*, 60439, 2014.

[10] Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

[11] Jimmy Ba Diederik P. Kingma. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015.

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

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

[14] Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, et al. Scaling exponents across parameterizations and optimizers. *arXiv preprint arXiv:2407.05872*, 2024.

[15] Roger Grosse and James Martens. A Kronecker-Factored Approximate Fisher Matrix for Convolution Layers. *arXiv 1602.01407*, 2016.

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

[17] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[18] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[19] Satoki Ishikawa and Ryo Karakida. On the parameterization of second-order optimization effective towards the infinite width. *arXiv preprint arXiv:2312.12226*, 2023.

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

[21] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[22] Priya Kasimbeg, Frank Schneider, Runa Eschenhagen, Juhan Bae, Chandramouli Shama Sastry, Mark Saroufim, BOYUAN FENG, Less Wright, Edward Z. Yang, Zachary Nado, Sourabh Medapati, Philipp Hennig, Michael Rabbat, and George E. Dahl. Accelerating neural network training: An analysis of the algoperf competition. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=CtM5xjRSfm`.

[23] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.

[24] 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.

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

[26] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[27] Depen Morwani, Itai Shapira, Nikhil Vyas, Eran Malach, Sham Kakade, and Lucas Janson. A new perspective on shampoo's preconditioner. *arXiv preprint arXiv:2406.17748*, 2024.

[28] Shikai Qiu, Atish Agarwala, Jeffrey Pennington, and Lechao Xiao. Scaling collapse reveals universal dynamics in compute-optimally trained neural networks. In *OPT 2024: Optimization for Machine Learning*.

[29] Shikai Qiu, Andres Potapczynski, Marc Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices. *arXiv preprint arXiv:2406.06248*, 2024.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI*, 2019.

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

[32] Utkarsh Sharma and Jared Kaplan. Scaling laws from the data manifold dimension. *Journal of Machine Learning Research*, 23(9):1–34, 2022.

[33] Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

[34] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

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

[36] Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. *arXiv preprint arXiv:2405.13698*, 2024.

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

[38] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

[39] Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling. *arXiv preprint arXiv:2409.15156*, 2024.

[40] Greg Yang and Edward J. Hu. Feature Learning in Infinite-Width Neural Networks. *International Conference on Machine Learning (ICML)*, 2021.

[41] Greg Yang and Etai Littwin. Tensor Programs IVb: Adaptive Optimization in the Infinite-Width Limit. *International Conference on Learning Representations (ICLR)*, 2023.

[42] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[43] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[44] Greg Yang, James B. Simon, and Jeremy Bernstein. A Spectral Condition for Feature Learning. *Preprint arXiv:2310.17813*, 2023.

[45] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Feature learning in infinite-depth neural networks. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limiations of our work in the appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: Our paper doesn't aim to prove for theorems. Our results can be understood in an intuitive way.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: In the appendix, we fully described our experiment setup. We will also release the code for reproducing the main experiments in the paper at the time of publication.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We used public dataset and public model architecture. We will also release the code for reproducing the main experiments in the paper at the time of publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We described experiment details in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Due to the expensive computational cost of training language models, we run each experiment only once. Additionally, our conclusions do not rely on comparing small differences in performance of specific experiments, but their overall scaling trends. Running for multiple seeds will not impact the conclusion.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We described it in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impact of this work in the Appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no risk of misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We used openly available code and datasets, and follow the license requirements.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper did not conduct experiments with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper did not conduct crowdsourcing or experiments with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: We didn't use LLM in a way impacting the core value of the research.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Update Rules for Second-Order Optimizer

This section presents the update rules for the optimizers considered in our work, for the sake of completeness and reproducibility. Let $W_t \in \mathbb{R}^{m\times n}$ denote the weight matrix of a layer at time t, and let $G_t \in \mathbb{R}^{m\times n}$ denote its gradient. We use $w_t \in \mathbb{R}^{mn}$ and $g_t \in \mathbb{R}^{mn}$ to represent the flattened versions of the weights and gradients, respectively. The hyperparameters include the learning rate (η), momentum coefficients (β), regularization factors (ϵ), and inverse exponents (e). Elementwise multiplication is denoted by $\odot$, and vector division is applied elementwise.

Adam [11] adjusts the magnitude of the first momentum by the second momentum. The update rule is:

$$g_t \leftarrow \nabla_w \mathcal{L}(w_{t-1}) \tag{13}$$
$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{14}$$
$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t \tag{15}$$
$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t} \tag{16}$$
$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t} \tag{17}$$
$$w_t \leftarrow w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \tag{18}$$

Shampoo [16] preconditions the gradient with a Kronecker-factored preconditioner. The update rule is:

$$M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t \tag{19}$$
$$L_t \leftarrow \beta_2 L_{t-1} + (1-\beta_2) G_t G_t^T \tag{20}$$
$$R_t \leftarrow \beta_2 R_{t-1} + (1-\beta_2) G_t^T G_t \tag{21}$$
$$\hat{L}_t \leftarrow \frac{L_t}{(1-\beta_2^t)} \tag{22}$$
$$\hat{R}_t \leftarrow \frac{R_t}{(1-\beta_2^t)} \tag{23}$$
$$W_{t+1} \leftarrow W_t - \eta (\hat{L}_t + \epsilon I)^{-e_L} M_t (\hat{R}_t + \epsilon I)^{-e_R} \tag{24}$$
$$\tag{25}$$

SOAP [35] runs Adam in a rotated space. The update rule is:

$$M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t \tag{26}$$
$$L_t \leftarrow \beta_2 L_{t-1} + (1-\beta_2) G_t G_t^T \tag{27}$$
$$R_t \leftarrow \beta_2 R_{t-1} + (1-\beta_2) G_t^T G_t \tag{28}$$
$$Q_L \leftarrow \text{Eigenvector}(L_t) \tag{29}$$
$$Q_R \leftarrow \text{Eigenvector}(R_t) \tag{30}$$
$$G_t' \leftarrow Q_L^T G_t Q_R \tag{31}$$
$$V_t \leftarrow \beta_2 V_{t-1} + (1-\beta_2) G_t' \odot G_t' \tag{32}$$
$$W_{t+1} \leftarrow W_t - \eta Q_L \frac{M_t}{\sqrt{V_t}+\epsilon} Q_R^T \tag{33}$$
$$\tag{34}$$

Muon [20] utilizes Newton-Schulz to compute the matrix sign [9] of the gradient. The update rule is:

$$M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t \tag{35}$$
$$W_{t+1} \leftarrow W_t - \eta \,\text{Newton-Schulz}\left(\frac{M_t}{\|M_t\|_2 + \epsilon}\right) \tag{36}$$
$$\tag{37}$$

Grafting [2, 33] is a technique that adjusts the direction of a primary optimizer's update to match the step size of a reference optimizer with more stable magnitude. Let ΔW_p denote the update from the primary optimizer, and ΔW_g the update from the reference (grafted) optimizer. The grafted update is given by:

$$W_{t+1} \leftarrow W_t - \eta \frac{\|\Delta W_p\|_2}{\|\Delta W_g\|_2 + \epsilon} \Delta W_g \tag{38}$$

Blocking [33, 35] is a technique that partitions the weight matrix into sub-blocks, and updates are computed independently for each block.

B Derivations of Maximum Update Parameterization

Moving forward, we will omit the layer subscripts and abbreviate $x(\xi)$ as x, $x(\xi')$ as x', $\delta(\xi)$ as δ, and $G(\xi)$ as G. To make the scaling with width explicit, we define the following normalized quantities[2]:

$$x' \equiv |x'\rangle\,, \quad x \equiv |x\rangle\,, \quad x^\top \equiv d_{\text{in}} \langle x|\,, \quad \delta \equiv \frac{1}{d_{\text{out}}} |\delta\rangle\,, \quad \delta^\top \equiv \langle \delta|\,. \tag{39}$$

The quantity $|z\rangle$ is referred to as a ket, representing a properly scaled version of z such that the entries are $\Theta(1)$, while $\langle z|$ is referred to as a bra, representing a properly scaled version of $z^\top$ such that $\langle z|z'\rangle = \Theta(1)$ for any z' correlated with z. Using this notation and the abbreviations, we can rewrite the condition for the learning rate as:

Choose learning rate η such that

$$\eta Q(G)\,|x'\rangle = \Theta(1), \text{where } G = \frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle\langle x|\,, \langle x|x'\rangle = \Theta(1). \tag{40}$$

Solving for the learning rate (and additional hyperparameters) now reduces to straightforward algebraic manipulations. As an example, we next show how to derive the scaling rules for the learning rate η and damping parameter ϵ in SOAP [35].

B.1 Warmup: SGD and Adam

As a warmup, we start with rederiving μPfor SGD and Adam under our notation.

SGD. We plug in $Q = \text{id}$ for SGD:

$$\eta Q(G)\,|x'\rangle = \eta \frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle \underbrace{\langle x|x'\rangle}_{\Theta(1)} = \Theta\left(\eta \frac{d_{\text{in}}}{d_{\text{out}}}\right). \tag{41}$$

Therefore we need $\eta = \Theta(\frac{d_{\text{out}}}{d_{\text{in}}})$, recovering results in Yang and Hu [40], Yang et al. [44]. Note that the use of bra-ket notation reduces the problem of finding the right learning rate into one straightforward algebraic manipulation.

SignSGD and Adam. For SignSGD, we have

$$Q(G) = \left(\sqrt{G^{\odot 2}} + \epsilon\right)^{-1} \odot G \tag{42}$$

$$= \left(\sqrt{\delta^{\odot 2} \otimes x^{\odot 2}} + \epsilon\right)^{-1} \odot \delta x^\top \tag{43}$$

$$= \left(\sqrt{\frac{1}{{d_{\text{out}}}^2} |\delta\rangle^{\odot 2} \otimes |x\rangle^{\odot 2}} + \epsilon\right)^{-1} \odot \frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle\langle x| \tag{44}$$

$$= d_{\text{in}} \left(\sqrt{|\delta\rangle^{\odot 2} \otimes |x\rangle^{\odot 2}} + \epsilon'\right)^{-1} \odot |\delta\rangle\langle x|\,, \tag{45}$$

[2]This notation is directly inspired by the bra-ket notation used in Yang and Littwin [41]. However, we use it strictly for keeping track of scaling factors rather than defining abstract infinite-width limits of gradients and activations in the network as scalar random variables.

where we reparameterized $\epsilon = \epsilon'/d_{\text{out}}$. For ϵ to be effective without trivializing the preconditioner, we need $\epsilon' = \Theta(1)$. Yang and Littwin [41] showed that applying a $\Theta(1)$ elementwise scaling $\left(\sqrt{|\delta\rangle^{\odot 2} \otimes |x\rangle^{\odot 2}} + \epsilon'\right)^{-1}$ to $|\delta\rangle\langle x|$ does not change the scale of its output when applied to $|x'\rangle$[3], so that $\Theta(Q(G)\,|x'\rangle) = \Theta(d_{\text{in}}\,|\delta\rangle\langle x|\,|x'\rangle) = \Theta(d_{\text{in}})$ and we should set $\eta = \Theta(1/d_{\text{in}})$.

The above analysis can be extended straightforwardly to Adam with the same conclusion $\eta = \Theta(1/d_{\text{in}})$ since the accumulation of the gradient over the iterates does not introduce extra scaling with width.

B.2 Second-Order Optimizers

With our setup so far, generalizing the derivation to second-order optimizers is straightforward, as one only needs to additional keep track of width scaling in the preconditioner. We will omit accumulation in the preconditioner and the momentum of the gradient, which does not affect the scaling with width [41].

K-FAC. In K-FAC, we have

$$\begin{aligned} Q(G) &= (\delta\delta^\top + \epsilon_B)^{-e_B}\delta x^\top (xx^\top + \epsilon_A)^{-e_A} \\ &= \left(\frac{1}{d_{\text{out}}}\,|\delta\rangle\langle\delta| + \epsilon_B\right)^{-e_B} |\delta\rangle\langle x|\,(d_{\text{in}}\,|x\rangle\langle x| + \epsilon_A)^{-e_A} \\ &= \frac{{d_{\text{in}}}^{1-e_A}}{{d_{\text{out}}}^{1-e_B}}(|\delta\rangle\langle\delta| + \epsilon'_B)^{-e_B}\,|\delta\rangle\langle x|\,(|x\rangle\langle x| + \epsilon'_A)^{-e_A}, \end{aligned} \tag{46}$$

where $\epsilon_B \equiv \epsilon'_B/d_{\text{out}}$ and $\epsilon_A \equiv d_{\text{in}}\epsilon'_A$. As $d \to \infty$, To ensure ϵ_A and ϵ_B have a consistent effect across widths and that each preconditioner does not trivialize to a scalar multiplication, we require $\epsilon'_B \sim \epsilon'_A \sim 1$. Using the identity

$$(vv^\top + aI)^{-e} = a^{-e}\left(I - \frac{vv^\top}{v^\top v}\right) + (v^\top v + a)^{-e}\frac{vv^\top}{v^\top v} \tag{47}$$

we can express the matrix inverses using only outer products and inner products as:

$$\begin{aligned} Q(G) = \frac{{d_{\text{in}}}^{1-e_A}}{{d_{\text{out}}}^{1-e_B}} &\times \left[\epsilon_B'^{-e_B}\left(I - \frac{|\delta\rangle\langle\delta|}{\langle\delta|\delta\rangle}\right) + (\langle\delta|\delta\rangle + \epsilon'_B)^{-e_B}\frac{|\delta\rangle\langle\delta|}{\langle\delta|\delta\rangle}\right] \\ &\times |\delta\rangle\langle x| \\ &\times \left[\epsilon_A'^{-e_A}\left(I - \frac{|x\rangle\langle x|}{\langle x|x\rangle}\right) + (\langle x|x\rangle + \epsilon'_A)^{-e_A}\frac{|x\rangle\langle x|}{\langle x|x\rangle}\right] \end{aligned} \tag{48}$$

This form makes it clear that 1) K-FAC can be expressed using existing instructions in the Tensor Program, a key result for establishing the existence of an infinite-width limit missing from prior work by Ishikawa and Karakida [19]

When the batch size is greater than 1, expressing the matrix inverse as outer and inner products is more involved but does not change our conclusion, which we explain in the next section.

[3]The output can be computed via an order-1 `OuterNonlin` instruction in Yang and Littwin [41], which has an $\Theta(1)$ limit.

Shampoo and Muon. The analysis for K-FAC can be straightforwardly adapted to Shampoo and Muon. Omitting accumulation and momentum, we have

$$\begin{aligned} Q_{\text{Shampoo}}(G) &= (\delta x^\top x \delta^\top + \epsilon_B)^{-e_L} \delta x^\top (x \delta^\top \delta x^\top + \epsilon_A)^{-e_R} \\ &= \left(\frac{d_{\text{in}}}{d_{\text{out}}}(\langle x|x\rangle\, |\delta\rangle\langle\delta| + \epsilon'_B)\right)^{-e_L} \\ &\quad \times \frac{d_{\text{in}}}{d_{\text{out}}}\, |\delta\rangle\langle x| \left(\frac{d_{\text{in}}}{d_{\text{out}}}(\langle\delta|\delta\rangle\, |x\rangle\langle x| + \epsilon'_A)\right)^{-e_R} \\ &= \left(\frac{d_{\text{in}}}{d_{\text{out}}}\right)^{1-e_L-e_R} (\langle x|x\rangle\, |\delta\rangle\langle\delta| + \epsilon'_B)^{-e_L} \\ &\quad \times |\delta\rangle\langle x|\, (\langle\delta|\delta\rangle\, |x\rangle\langle x| + \epsilon'_A)^{-e_R} \end{aligned} \tag{49}$$

with $\epsilon'_{A,B} \equiv \frac{d_{\text{out}}}{d_{\text{in}}}\epsilon_{A,B}$. Expressing the matrix inverses as inner and outer products:

$$\begin{aligned} Q_{\text{Shampoo}}(G) = &\left(\frac{d_{\text{in}}}{d_{\text{out}}}\right)^{1-e_L-e_R} \\ &\times \left[\epsilon'^{-e_L}_B \left(I - \frac{|\delta\rangle\langle\delta|}{\langle\delta|\delta\rangle}\right) + (\langle x|x\rangle\, \langle\delta|\delta\rangle + \epsilon'_B)^{-e_L} \frac{|\delta\rangle\langle\delta|}{\langle\delta|\delta\rangle}\right] \\ &\times |\delta\rangle\langle x| \\ &\times \left[\epsilon'^{-e_R}_A \left(I - \frac{|x\rangle\langle x|}{\langle x|x\rangle}\right) + (\langle x|x\rangle\, \langle\delta|\delta\rangle + \epsilon'_A)^{-e_R} \frac{|x\rangle\langle x|}{\langle x|x\rangle}\right]. \end{aligned} \tag{50}$$

Therefore Shampoo and Muon can also be expressed using existing Tensor Program instructions. Furthermore, we can see that $Q_{\text{Shampoo}}(G)\,|x'\rangle = \Theta\left(\frac{d_{\text{in}}}{d_{\text{out}}}\right)^{1-e_L-e_R} |\delta\rangle$, implying that we need $\eta = \Theta\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)^{1-e_L-e_R}$ for proper scaling.

SOAP. SOAP runs the Adam optimizer in the eigenbasis of Shampoo's preconditioner. Ignoring momentum, we have

$$G' = U^\top G V \tag{51}$$

$$Q(G) = U\left[\left(\sqrt{G'^{\odot 2}} + \epsilon\right)^{-1} \odot G'\right] V^\top \tag{52}$$

where U and V are the eigenbases of the left (L) and right (R) preconditioners from Shampoo. For the single-batch case, these matrices are

$$L = \delta x^\top x \delta^\top + \epsilon_B = \frac{d_{\text{in}}}{d_{\text{out}}}(\langle x|x\rangle\, |\delta\rangle\langle\delta| + \epsilon'_B) \tag{53}$$

$$R = x \delta^\top \delta x^\top + \epsilon_A = \frac{d_{\text{in}}}{d_{\text{out}}}(\langle\delta|\delta\rangle\, |x\rangle\langle x| + \epsilon'_A) \tag{54}$$

with $\epsilon'_{A,B} = \frac{d_{\text{out}}}{d_{\text{in}}}\epsilon_{A,B}$.

The matrix L has eigenvalue $\frac{d_{\text{in}}}{d_{\text{out}}}\langle x|x\rangle\langle\delta|\delta\rangle + \epsilon_B$ with eigenvector in the direction of $|\delta\rangle$, and eigenvalue ϵ_B with multiplicity $d_{\text{out}} - 1$ for the orthogonal subspace. Similarly, R has eigenvalue $\frac{d_{\text{in}}}{d_{\text{out}}}\langle\delta|\delta\rangle\langle x|x\rangle + \epsilon_A$ with eigenvector in the direction of $|x\rangle$, and eigenvalue ϵ_A with multiplicity $d_{\text{in}} - 1$ for the orthogonal subspace.

Hence, U is a matrix with the first column $\frac{|\delta\rangle}{\sqrt{d_{\text{out}}\langle\delta|\delta\rangle}}$ and the remaining columns forming an orthonormal basis for the space orthogonal to $|\delta\rangle$. Similarly, V is a matrix with first column $\frac{|x\rangle}{\sqrt{d_{\text{in}}\langle x|x\rangle}}$ and remaining columns forming an orthonormal basis for the space orthogonal to $|x\rangle$.

Transforming the gradient to the eigenbasis:

$$G' = U^\top G V \tag{55}$$

$$= U^\top \left(\frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle\langle x| \right) V \tag{56}$$

$$= \frac{|\delta\rangle^\top}{\sqrt{d_{\text{out}} \langle\delta|\delta\rangle}} \left(\frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle\langle x| \right) \frac{|x\rangle}{\sqrt{d_{\text{in}} \langle x|x\rangle}} e_1^U {e_1^V}^\top \tag{57}$$

$$= \frac{\sqrt{d_{\text{out}}} \langle\delta|}{\sqrt{\langle\delta|\delta\rangle}} \left(\frac{d_{\text{in}}}{d_{\text{out}}} |\delta\rangle\langle x| \right) \frac{|x\rangle}{\sqrt{d_{\text{in}} \langle x|x\rangle}} e_1^U {e_1^V}^\top \tag{58}$$

$$= \sqrt{\frac{d_{\text{in}} \langle\delta|\delta\rangle \langle x|x\rangle}{d_{\text{out}}}} e_1^U {e_1^V}^\top \tag{59}$$

where e_1^U and e_1^V are the first standard basis vectors in the dimensions of U and V, respectively.

For ϵ to be effective without trivializing the preconditioner, we need it to be on the same scale as G', so $\epsilon = \Theta\left(\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\right) = \sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\epsilon'$ where $\epsilon' = \Theta(1)$. The Adam-like operation in the eigenbasis then gives:

$$\left(\sqrt{G'^{\odot 2}} + \epsilon\right)^{-1} \odot G' = \frac{1}{1+\epsilon'} e_1^U {e_1^V}^\top \tag{60}$$

Transforming back to the original basis:

$$Q_{\text{SOAP}}(G) = U\left[\left(\sqrt{G'^{\odot 2}} + \epsilon\right)^{-1} \odot G'\right] V^\top \tag{61}$$

$$= \frac{1}{1+\epsilon'} \frac{|\delta\rangle}{\sqrt{d_{\text{out}} \langle\delta|\delta\rangle}} \frac{d_{\text{in}} \langle x|}{\sqrt{d_{\text{in}} \langle x|x\rangle}} \tag{62}$$

$$= \frac{1}{1+\epsilon'} \frac{d_{\text{in}} |\delta\rangle\langle x|}{\sqrt{d_{\text{in}} d_{\text{out}} \langle\delta|\delta\rangle \langle x|x\rangle}} \tag{63}$$

To determine the proper learning rate, we apply the condition $\eta Q_{\text{SOAP}}(G)|x'\rangle = \Theta(1)$:

$$Q_{\text{SOAP}}(G)|x'\rangle = \frac{1}{1+\epsilon'} \frac{d_{\text{in}} |\delta\rangle \langle x|x'\rangle}{\sqrt{d_{\text{in}} d_{\text{out}} \langle\delta|\delta\rangle \langle x|x\rangle}} \tag{64}$$

$$= \frac{1}{1+\epsilon'} \frac{d_{\text{in}} |\delta\rangle \langle x|x'\rangle}{\sqrt{d_{\text{in}} d_{\text{out}} \langle\delta|\delta\rangle \langle x|x\rangle}} \tag{65}$$

Since $\langle x|x'\rangle = \Theta(1)$, $\langle\delta|\delta\rangle = \Theta(1)$, and $\langle x|x\rangle = \Theta(1)$, we have:

$$Q_{\text{SOAP}}(G)|x'\rangle = \Theta\left(\frac{d_{\text{in}}}{\sqrt{d_{\text{in}} d_{\text{out}}}}\right)|\delta\rangle = \Theta\left(\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\right)|\delta\rangle \tag{66}$$

Therefore, the scaling is $\eta Q_{\text{SOAP}}(G)|x'\rangle = \Theta\left(\eta\sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}}\right)$, implying we need $\eta = \Theta\left(\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}\right)$ for SOAP.

Blocking. Blocking is a technique to reduce the preconditioner cost in second-order optimizers like Shampoo by subdividing each weight matrix into $b_{\text{out}} \times b_{\text{in}}$ blocks and preconditioning each block separately. We derive μP for Shampoo with Blocking but generalizing to other optimizers can be done with a similar analysis. We simply perform the same analysis for Shampoo inside each block and appropriately aggregate their contribution to the feature update. Let $G_{ij} = \delta_i x_j^\top \in \mathbb{R}^{b_{\text{out}} \times b_{\text{in}}}$ index the ij-th block of the gradient, where $\delta_i \in \mathbb{R}^{b_{\text{out}}}$ and $x_j \in \mathbb{R}^{b_{\text{in}}}$ are the i and j-th chunk of the vector δ and x. Similarly, let $x'_j \in \mathbb{R}^{b_{\text{in}}}$ be the j-th chunk of the input vector x'. Defining the sub-kets and sub-bras

$$x'_j \equiv |x'_j\rangle, \quad x_j \equiv |x_j\rangle, \quad x_j^\top \equiv b_{\text{in}} \langle x_j|, \tag{67}$$

$$\delta_i \equiv \frac{1}{d_{\text{out}}} |\delta_i\rangle, \quad \delta_i^\top \equiv \frac{b_{\text{out}}}{d_{\text{out}}} \langle\delta_i|. \tag{68}$$

The i-th chunk of the update Δh_i is

$$\Delta h_i = \eta \sum_{j=1}^{d_{\text{in}}/b_{\text{in}}} Q(G_{ij}) \left|x'_j\right\rangle, \tag{69}$$

where

$$\begin{aligned} Q(G_{ij}) &= (\delta_i x_j^\top x_j \delta_i^\top + \epsilon_B)^{-e_L} \delta_i x_j^\top (x_j \delta_i^\top \delta_i x_j^\top + \epsilon_A)^{-e_R} \\ &= \left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2} (\langle x_j | x_j \rangle \left|\delta_i\right\rangle\!\left\langle\delta_i\right| + \epsilon'_B)\right)^{-e_L} \\ &\quad \times \frac{b_{\text{in}}}{d_{\text{out}}} \left|\delta_i\right\rangle\!\left\langle x_j\right| \left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2} (\langle \delta_i | \delta_i \rangle \left|x_j\right\rangle\!\left\langle x_j\right| + \epsilon'_A)\right)^{-e_R} \\ &= \left(\frac{b_{\text{in}}}{d_{\text{out}}}\right)\left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2}\right)^{-(e_L+e_R)} (\langle x_j | x_j \rangle \left|\delta_i\right\rangle\!\left\langle\delta_i\right| + \epsilon'_B)^{-e_L} \\ &\quad \times \left|\delta_i\right\rangle\!\left\langle x_j\right| (\langle \delta_i | \delta_i \rangle \left|x_j\right\rangle\!\left\langle x_j\right| + \epsilon'_A)^{-e_R}, \end{aligned} \tag{70}$$

with $\epsilon_{A,B} \equiv \frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2} \epsilon'_{A,B}$. Therefore,

$$\Delta h_i = \eta \left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2}\right)^{-(e_L+e_R)} \left(\frac{b_{\text{in}}}{d_{\text{out}}}\right) \sum_{j=1}^{d_{\text{in}}/b_{\text{in}}} \Theta(1) \left|\delta_i\right\rangle \tag{71}$$

$$= \eta \left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2}\right)^{-(e_L+e_R)} \left(\frac{b_{\text{in}}}{d_{\text{out}}}\right) \Theta\left(\frac{d_{\text{in}}}{b_{\text{in}}}\right) \left|\delta_i\right\rangle \tag{72}$$

$$= \Theta\left(\eta \left(\frac{d_{\text{in}}}{d_{\text{out}}}\right)\left(\frac{b_{\text{in}} b_{\text{out}}}{{d_{\text{out}}}^2}\right)^{-(e_L+e_R)}\right), \tag{73}$$

where we used the fact that each term in the sum is i.i.d. with a (generally) non-zero mean. Therefore the learning rate should scale as $\eta = \Theta\left(\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)\left(\frac{{d_{\text{out}}}^2}{b_{\text{in}} b_{\text{out}}}\right)^{-(e_L+e_R)}\right)$. Alternatively, let $n_{\text{in}} = d_{\text{in}}/b_{\text{in}}$ and $n_{\text{out}} = d_{\text{out}}/b_{\text{out}}$, we have $\eta = \Theta\left(\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)^{1-(e_L+e_R)} (n_{\text{in}} n_{\text{out}})^{-(e_L+e_R)}\right)$. When $b_{\text{in}} = d_{\text{in}}$ and $b_{\text{out}} = d_{\text{out}}$, we recover the regular Shampoo scaling $\eta = \Theta\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)^{1-e_L-e_R}$. When $e_L + e_R = \frac{1}{2}$ and $b_{\text{in}} = b_{\text{out}} = 1$, degenerating to Adam, we get the correct scaling $\eta = \Theta\left(\frac{1}{d_{\text{in}}}\right)$.

Normalization by Frobenius Norm. Consider normalizing the preconditioned gradient by its Frobenius Norm:

$$Q(G) = \frac{Q_1(G)}{\|Q_1(G)\|_F + \epsilon} \tag{74}$$

$$= \frac{Q_1(G)}{\sqrt{\text{Tr}(Q_1(G)^\top Q_1(G))} + \epsilon}. \tag{75}$$

Let $Q_1(G) = q\hat{G}_1$ where q is a scalar that absorbs all width-dependent scaling and $\hat{G}_1$ contains only bras and kets. Then $Q_1(G)^\top = q\hat{G}_1^\top = q\frac{d_{\text{out}}}{d_{\text{in}}}\hat{G}_1^\dagger$, where $X^\dagger$ is $X^\top$ but with all vectors and their transposes replaced with kets and bras. Therefore, we get

$$Q(G) = \frac{q_1 \hat{G}_1}{\sqrt{q_1^2 \frac{d_{\text{out}}}{d_{\text{in}}} \text{Tr}\left(\hat{G}_1^\dagger \hat{G}_1\right)} + \epsilon} \tag{76}$$

$$= \frac{\hat{G}_1}{\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)^{1/2} \left(\text{Tr}\left(\hat{G}_1^\dagger \hat{G}_1\right)^{1/2} + \frac{\epsilon}{q_1 (d_{\text{out}}/d_{\text{in}})^{1/2}}\right)}. \tag{77}$$

We should choose $\epsilon \sim q_1\sqrt{d_{\text{out}}/d_{\text{in}}}$, and $\eta \sim \sqrt{d_{\text{out}}/d_{\text{in}}}$.

Grafting. Grafting combines the direction of the update from one optimizer with preconditioner Q_1 with the magnitude of the update from another with preconditioner Q_2, with the final update given by:

$$Q(G) = \|Q_2(G)\|_F \frac{Q_1(G)}{\|Q_1(G)\|_F + \epsilon} \tag{78}$$

$$= q_2 \operatorname{Tr}\left(\hat{G}_2^\dagger \hat{G}_2\right)^{1/2} \frac{\hat{G}_1}{\operatorname{Tr}\left(\hat{G}_1^\dagger \hat{G}_1\right)^{1/2} + \frac{\epsilon}{q_1 (d_{\text{out}}/d_{\text{in}})^{1/2}}}. \tag{79}$$

For proper scaling, we need $\epsilon \sim q_1\sqrt{d_{\text{out}}/d_{\text{in}}}$ and $\eta \sim 1/q_2$, i.e. the learning rate should scale as if we are training directly with Q_2.

Depth-scaling. As explained in the main text, we scale down the output of each residual block by $1/L$ while ensuring $\Theta(1)$ feature learning inside each block following [8, 10] which showed this criterion led to well-defined depth-scaling limits for residual networks like the transformer. The only change to the previous width-scaling derivation is that gradients in the residual blocks now scale as $\frac{1}{Ld_{\text{out}}}$ rather than $\frac{1}{d_{\text{out}}}$. Therefore, we need to update the definition of $|\delta\rangle$ and $\langle\delta|$ to $\delta = \frac{1}{Ld_{\text{out}}}|\delta\rangle$ and $\delta^\top = \frac{1}{L}\langle\delta|$. Repeating the above derivations while accounting for these additional L-dependent factors straightforwardly leads to the width-depth joint scaling rules in Table 1. We include the derivation for Shampoo here as a concrete example:

$$\begin{aligned}
Q_{\text{Shampoo}}(G) &= (\delta x^\top x \delta^\top + \epsilon_B)^{-e_L} \delta x^\top (x\delta^\top \delta x^\top + \epsilon_A)^{-e_R} \\
&= \left(\frac{d_{\text{in}}}{L^2 d_{\text{out}}}(\langle x|x\rangle\,|\delta\rangle\langle\delta| + \epsilon'_B)\right)^{-e_L} \\
&\quad \times \frac{d_{\text{in}}}{Ld_{\text{out}}}\,|\delta\rangle\langle x| \left(\frac{d_{\text{in}}}{L^2 d_{\text{out}}}(\langle\delta|\delta\rangle\,|x\rangle\langle x| + \epsilon'_A)\right)^{-e_R} \\
&= \left(\frac{d_{\text{in}}}{d_{\text{out}}}\right)^{1-e_L-e_R} \left(\frac{1}{L}\right)^{1-2(e_L+e_R)} \\
&\quad \times \underbrace{(\langle x|x\rangle\,|\delta\rangle\langle\delta| + \epsilon'_B)^{-e_L}\,|\delta\rangle\langle x|\,(\langle\delta|\delta\rangle\,|x\rangle\langle x| + \epsilon'_A)^{-e_R}}_{\Theta(1)\text{ as before}}
\end{aligned} \tag{80}$$

with $\epsilon'_{A,B} \equiv \frac{L^2 d_{\text{out}}}{d_{\text{in}}}\epsilon_{A,B}$. Therefore we need $\eta \sim L^{1-2(e_L+e_R)}\left(\frac{d_{\text{out}}}{d_{\text{in}}}\right)^{1-e_L-e_R}$ and $\epsilon \sim \frac{d_{\text{in}}}{L^2 d_{\text{out}}}$.

C Extension to Larger Batch Size

Throughout the main text we analysed single-sample updates ($B = 1$). All scaling results survive *unchanged* for any *constant* mini-batch size $B = \Theta(1)$ because every object that appears in the update rules becomes only a finite sum of the same single-sample terms.

SGD. With batch size B the raw update is

$$\Delta W \;=\; -\frac{\eta}{B}\sum_{b=1}^{B} \underbrace{\delta^{(b)}\, x^{(b)^\top}}_{G^{(b)}},$$

where each $G^{(b)}$ has entries $\Theta\big(1/d_{\text{out}}\big)$ (see §3.1). Averaging over the *finite* set $\{1, \ldots, B\}$ do not alter the scaling of the update with width, so the condition $\Delta W\, x' = \Theta(1)$ is met with the *same* learning-rate scaling $\eta = \Theta(d_{\text{out}}/d_{\text{in}})$ derived for $B = 1$.

Adam. Adam first rescales the averaged gradient by the element-wise second moment v:

$$\Delta W \;=\; -\,\eta \left(\frac{1}{B}\sum_{b=1}^{B} G^{(b)}\right) \oslash \big(\sqrt{v} + \epsilon\big),$$

Optimizer	Width	LR	β_2	Adam LR. mult.	Warmup	Indep. WD	Loss
Adam	512	4e−3	0.98	—	20%	2e−4	3.236
Muon	512	8e−3	—	1.6	0.3%	3.2e−4	3.178
Muon	768	—	—	1.6	0.3%	3.2e−4	3.157

Table 3: Best validation loss for base model after hyperparameter and token-per-parameter tuning.

Optimizer	Width	Loss	TPP
Adam	576	3.226039	13.338798
Adam	896	2.996272	11.494327
Adam	1152	2.855488	12.975372
Muon	704	3.157175	6.607409
Muon	1024	2.943154	7.071705
Muon	1344	2.805800	7.325234

Table 4: **Optimal widths for the compute budgets of the** 190**M,** 380**M, and** 640**M models**. Muon has a $2\times$ smaller optimal TPP than Adam.

with $v = \beta_2 v_{\text{old}} + (1-\beta_2)\frac{1}{B}\sum_b (G^{(b)})^2$. Again since $B = \Theta(1)$, compared to the $B = 1$ update, both the numerator and denominator changes by only a $\Theta(1)$ factor, giving the same width-scaling $\eta = \Theta(1/d_{\text{in}})$.

Shampoo and related preconditioners. A common operation in second-order optimizers have the form $Q(G) = L^{-e_L} G R^{-e_R}$, where the L and R are left/right Gram matrices of the mini-batch gradient (damped by ϵ_L, ϵ_R as in Appendix A). Because B is $\Theta(1)$, each matrix is a sum of finitely many rank-1 outer products of $|\delta\rangle$ or $|x\rangle$; hence every eigenvalue of L and R scales identically to the $B = 1$ case, and so are the eigenvalues of L^{-e_L} and R^{-e_R}. Consequently the learning-rate and regularization rules in Table 1 require no modification.

In short, as long as the mini-batch size stays fixed with respect to width, every factor introduced by batching is $\Theta(1)$; therefore the scaling derived for $B = 1$ extends unchanged to all $B = \Theta(1)$.

D Experiment Details

D.1 Hyperparameter Tuning Details

Adam. Phase 1 sweeps the triplet {learning rate, β_2, warmup steps}. The best setting is lr $= 0.004$, $\beta_2 = 0.98$, and a warmup of 20% of total training steps. Holding β_2 and warmup fixed, Phase 2 jointly sweeps {learning rate, weight decay}, yielding lr $= 0.004$ and weight decay $= 0.002$. The best loss achieved with Adam is 3.236.

Muon-Adam. Phase 1 sweeps {learning rate, Adam LR multiplier, warmup tokens}. The best setting is lr $= 0.008$, multiplier $= 1.6$, and a warmup of 0.003% of total training tokens. With the multiplier and warmup fixed, Phase 2 jointly sweeps {learning rate, weight decay}, selecting lr $= 0.008$ and weight decay $= 3.2 \times 10^{-4}$. The best loss is 3.178.

The best validation losses and the corresponding hyperparameters are listed in Table 3

Tokens-per-Parameter (TPP). We tune TPP by holding total FLOPs fixed and varying model width. We find that if we scale the learning rate using μP rule and fix the product of learning rate and independent weight decay (i.e. dependent weight decay) as constant, the weight decay and learning rate are optimal. Therefore, we adopt this scaling rule to find the optimal TPP without tuning hyperparameters. The optimal TPP are shown in Figure 8 and Table 4. To set the compute-optimal scaling experiments for the 1.4B model. We fix the compute budget and choose the model width so that the corresponding TPP is closest to what we found in Table 4. Adam uses width 1728 with TPP 12.83 and Muon uses width 2048 with TPP 6.73.

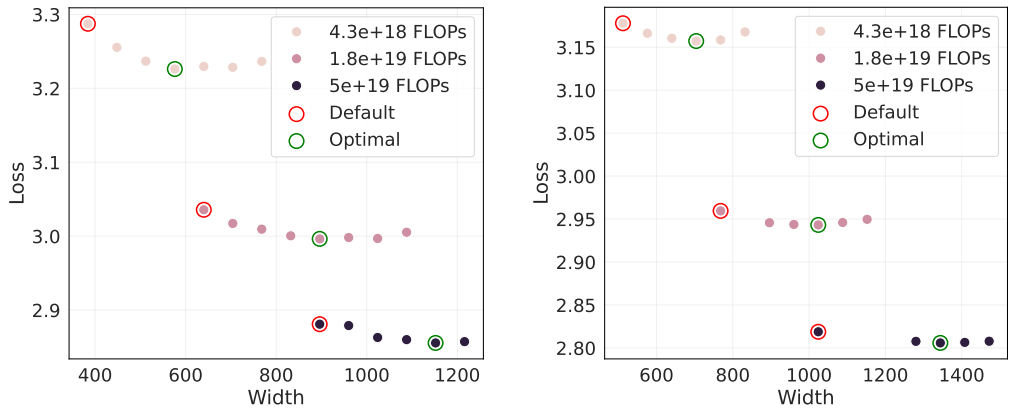

Figure 8: **Tuning token-per-parameter for Adam (left) and Muon (right)**. Both optimizers have a different optimal TPP from the default constant 20 from Hoffmann et al. [18]. Muon's optimal TPP is further from the default constant than Adam's.

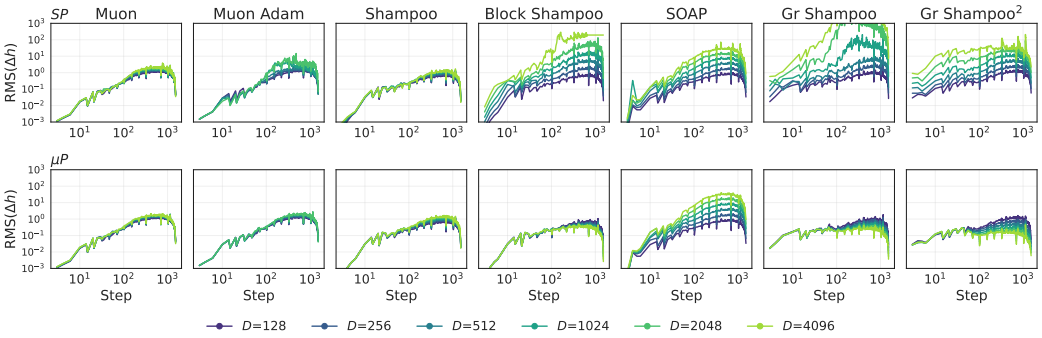

Figure 9: Feature changes for OpenWebText. Our scaling approach ensures a stable feature changes over widths whereas the naive approach doesn't.

D.2 OpenWebText Experiments

For experiments conducted on OpenWebText, we used character-level tokenization to reduce the cost of the embedding layers. Not doing so leads total parameter count being dominated by embedding parameters at small scales, which has been found to lead to distorted scaling laws for small models [29, 21]. We use the GPT-2 architecture and follow common practices of removing the bias and replacing LayerNorm with RMSNorm. Learning rates are tuned on a small 3 layer width 128 model and transferred to larger models. Unless stated otherwise, we use a batch size of 65536 tokens and linearly warmup the learning rate in the first $10M$ tokens. Except for compute-optimal scaling experiments, we train all models for 1B tokens with a linearly decayed learning rate schedule.

RMS change of features. Stable feature changes across model widths are frequently used as proxies for verifying correct feature learning scaling [43, 19]. We provide the feature changes in Figure 9 as a proof for the correctness of our scaling approach.

Tokens-per-Parameter (TPP) To verify the prediction that Muon requires less tokens to become compute-optimal, we train a series of transformer models with different optimizers on OpenWebText, jointly scaling width and depth with a width-depth ratio of 64 and scaling hyperparameters according to our scaling rules using a fixed batch size of 512 sequences of length 128 each. In Figure 10 (left), we plot the optimal training tokens as a function of parameters using Approach 1 in Hoffmann et al. [18] for Adam, Muon, and Shampoo. Indeed, we find Muon and Shampoo require $\sim 2.5\times$ fewer tokens than Adam to be compute-optimal for the same model size. We show a similar multiplicative reduction in compute-optimal tokens for transformers trained on Chess moves from the Lichess dataset in Figure 10 (right), showing this result is not specific to language.

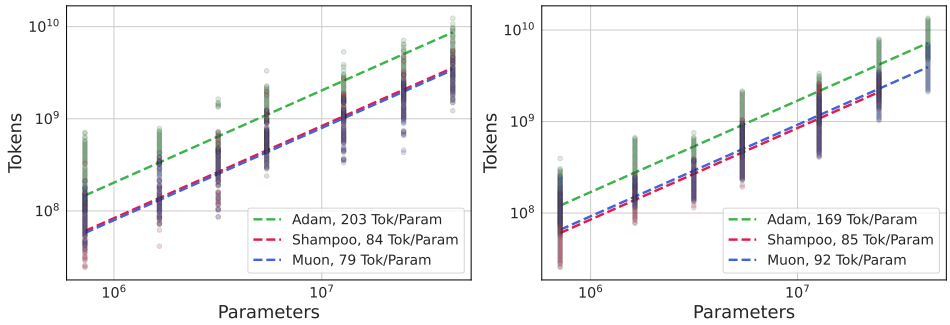

Figure 10: **Muon and Shampoo have up to $\sim 2\times$ smaller optimal token-per-parameter ratio than Adam.** (Left) OpenWebText. (Right) Lichess.

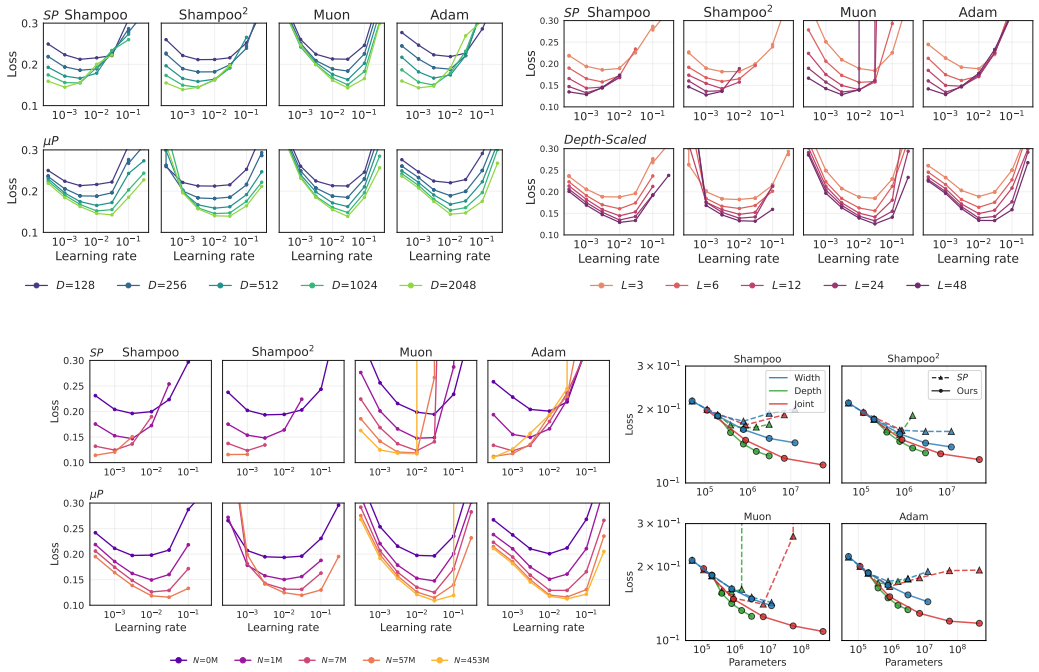

Figure 11: **Learning rate transfer in MLP.** Our proposal scaling rules lead to significantly more stable optimal learning rate for width-only scaling (top left), depth-only scaling (top right), and joint scaling (bottom left) for the residual MLP architecture described by Eq (14). Compared to the transformer, properly scaling the learning rate is more important for effective scaling of MLPs, achieving lower loss compared to SP using a constant learning rate (bottom right).

D.3 MLP Experiments

To better demonstrate the impact of proper learning rate scaling, we perform experiments with the residual MLP architecture described by Eq (14), known to be more sensitive to the learning rate than the transformer architecture [6]. We use the power-law Fourier features regression task [28] as it allows generating unlimited training data to avoid overfitting. All models are trained with 100M examples with a batch size of 4096. Figure 11 summarizes the results, showing our proposed scaling rules are crucial for effectively scaling the MLP models, stabilizing the optimal learning rate across model size and improving final performance as the model increases in size.

D.4 Hardware

We trained all of our models on TPU-V4 and TPU-V6e, supported by the Google TPU Research Cloud program.

E Broader Impact and Limitations

Broader Impact. Our work improves the understanding of how to efficiently scale second-order optimization in deep learning, which has the potential to reduce the cost of training machine learning models and make machine learning research and workflow more accessible.

Limitation. We perceive two main limitations of our work. First, due to limited computational budget, our experiments are relatively small in scale compared to typical training runs in industry where models often exceed billions of parameters. Verifying how well our results generalizes to larger models trained with more compute is an exciting future direction. Second, while we have shown that second-order optimizers like Shampoo and Muon can significantly improve the compute efficiency of training, we do not investigate why they improve the efficiency. Understanding this question holds potential for designing even more efficient future optimizers.