Scaling Exponents Across Parameterizations and Optimizers

Katie Everett [1 2] **Lechao Xiao** [1] **Mitchell Wortsman** [3] **Alexander A. Alemi** [1] **Roman Novak** [3] **Peter J. Liu** [1]
Izzeddin Gur [1] **Jascha Sohl-Dickstein** [3] **Leslie Pack Kaelbling** [2] **Jaehoon Lee** [1] **Jeffrey Pennington** [1]

Abstract

Robust and effective scaling of models from small to large width typically requires the precise adjustment of many algorithmic and architectural details, such as parameterization and optimizer choices. In this work, we propose a new perspective on parameterization by investigating a key assumption in prior work about the alignment between parameters and data and derive new theoretical results under weaker assumptions and a broader set of optimizers. Our extensive empirical investigation includes *tens of thousands* of models trained with *all combinations of* three optimizers, four parameterizations, several alignment assumptions, more than a dozen learning rates, and fourteen model sizes up to 26.8B parameters. We find that the best learning rate scaling prescription would often have been excluded by the assumptions in prior work. Our results show that all parameterizations, not just maximal update parameterization (muP), can achieve hyperparameter transfer; moreover, our novel per-layer learning rate prescription for standard parameterization outperforms muP. Finally, we demonstrate that an overlooked aspect of parameterization, the epsilon parameter in Adam, must be scaled correctly to avoid gradient underflow and propose *Adam-atan2*, a new numerically stable, scale-invariant version of Adam that eliminates the epsilon hyperparameter entirely.

1. Introduction

A neural network parameterization is a prescription for scaling a set of important quantities with respect to a set of scaling dimensions. Most often, the parameterized quantities include the initialization scale, parameter multipliers and learning rate, and scaling dimensions may include model width, model depth, context length, batch size and training horizon. Among the scaling dimensions, the existing literature for *width scaling* has the most extensive theoretical results. Yang & Hu (2021) and Yang & Littwin (2023) define a space of width-scaling parameterizations for stochastic gradient descent (SGD) and Adam respectively. Based on the goals of ensuring the activations remain at constant scale and the logits do not exceed constant scale with respect to width, they derive constraints for stability and non-triviality of a parameterization that predict how the learning rate should scale with width.

The derivations of these constraints make a key assumption: the updates to the parameters are assumed to be sufficiently correlated with the data distribution to impact the scaling exponent of the activations. We refer to this correlation as "alignment" because, when correlated, the parameter and data vectors point in similar directions. However, the literature is lacking in extensive empirical measurements of when, where, and how much alignment accumulates between the parameters and activations during training and its impact on the scaling exponents of the learning rate. Moreover, the characterization of parameterizations into feature learning and kernel limits are specific to the alignment assumptions.

In this paper, we take a broad perspective on the theory and practice of width scaling across parameterizations and optimizers. Our theoretical contributions consider a more general space of parameterizations which explicitly quantify the contribution of several distinct alignment terms; under specific alignment assumptions we recover prior work as a special case. We propose a metric for alignment that we use in our empirical investigation. In addition, we develop new parameterization theory for a family of adaptive optimizers with parameter scaling, including Adafactor.

In our experiments, we measure alignment throughout training across optimizers, parameterizations and model sizes. Our measurements suggest that existing theory may be overly conservative, thereby excluding interesting parameterizations. We show that for all parameterizations and optimizers, there are theoretically motivated per-layer learning rate exponents that improve performance over global learning rate baselines, and in numerous settings the best performing exponents would have been excluded by the

[1]Google DeepMind [2]MIT [3]Work done at Google DeepMind. Correspondence to: Katie Everett <everettk@google.com>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Table 1. Left: Parameterizations and gradients at initialization for width n. Middle: Max stable per-layer learning rate scaling for each optimizer assuming $\alpha_l = 1, \omega_l = 1/2$ for all layers l. Right: Max stable learning rates assuming $\alpha_l = \omega_l = u_l = 1/2$ for all layers.

		Initialization Variance	Parameter Multiplier	Gradient	SGD LR, Full Align	Adam LR, Full Align	Adafactor LR, Full Align	SGD LR, No Align	Adam LR, No Align	Adafactor LR, No Align
Standard	Embedding	1	1	$1/\sqrt{n}$	$\sqrt{n}$	1	1	$\sqrt{n}$	1	1
	Hidden	$1/n$	1	$1/\sqrt{n}$	$1/\sqrt{n}$	$1/n$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1
	Readout	$1/n$	1	1	$1/n$	$1/n$	$1/\sqrt{n}$	$1/\sqrt{n}$	$1/\sqrt{n}$	1
NTK	Embedding	1	1	$1/\sqrt{n}$	$\sqrt{n}$	1	1	$\sqrt{n}$	1	1
	Hidden	1	$1/\sqrt{n}$	$1/n$	$\sqrt{n}$	$1/\sqrt{n}$	$1/\sqrt{n}$	n	1	1
	Readout	1	$1/\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	$1/\sqrt{n}$	$\sqrt{n}$	1	1
muP	Embedding	$1/n$	$\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1	1	$1/\sqrt{n}$	1
	Hidden	$1/n$	1	$1/n$	1	$1/n$	$1/\sqrt{n}$	$\sqrt{n}$	$1/\sqrt{n}$	1
	Readout	$1/n$	$1/\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1	1	1	1
MFP	Embedding	1	1	$1/n$	n	1	1	n	1	1
	Hidden	1	$1/\sqrt{n}$	$1/n^{1.5}$	n	$1/\sqrt{n}$	$1/\sqrt{n}$	$n^{1.5}$	1	1
	Readout	1	$1/n$	$1/n$	n	1	1	n	$\sqrt{n}$	1

alignment assumptions of prior work. In particular, a novel per-layer learning rate prescription for standard parameterization is shown to outperform muP. In addition, while prior work emphasizes the hyperparameter transfer properties of muP specifically, we show that all parameterizations can perform hyperparameter transfer. We introduce constant multiplicative factors to the per-layer learning rate prescriptions and show that tuning these factors is both essential and practical: the constants can be tuned at relatively small scale and successfully reused across model sizes up to 26.8 billion parameters, inducing substantial performance gains.

Finally, we consider the epsilon hyperparameter in Adam and similar adaptive optimizers. Our theoretical prediction that the mean-field parameterization will be most sensitive to epsilon underflow is validated in our experiments. We see significant performance improvements from three mitigations to epsilon, including our proposal *Adam-atan2* that eliminates epsilon entirely. After addressing epsilon, parameterizations that are theoretically equivalent to give very similar performance, illustrating that finite precision plays a practical role in the study of parameterization.

2. Background

2.1. Parameterizations and Optimizers

We define a width-scaling parameterization as in Yang & Hu (2021) as the prescription of the scaling exponents for three quantities on each layer: (1) the initialization variance for the parameters, (2) a parameter multiplier[1] by which the trainable parameter weights are multiplied during the forward pass, and (3) the learning rate. It is typical for different layer types (embedding, hidden, and readout) to use different parameterizations within the same network.

[1]The parameter multiplier is a constant that multiplies the output of the matrix multiplication in the layer during the forward pass. The trainable parameters are updated during training but the parameter multiplier is not. However, the backpropagated gradients for the parameters will include this multiplier as a term.

We will consider all combinations of four common parameterizations and three common optimizer families. Our parameterizations, shown in Table 1, include standard (Neal, 1996; Glorot & Bengio, 2010; He et al., 2015), Neural Tangent Kernel (NTK) (Jacot et al., 2018), Maximal Update (muP) (Yang & Hu, 2021), and Mean Field (Bordelon & Pehlevan, 2022). Following convention, the *names* of parameterizations will refer to the initialization scale and parameter multipliers, although formally speaking, the learning rate prescription is an essential element of a parameterization.

We select optimizers that represent three distinct width-scaling regimes: Stochastic Gradient Descent (SGD), Adam (Kingma & Ba, 2014), and Adafactor (Shazeer & Stern, 2018) due to their varying relationships between the parameter, gradient, and update scales. Our theoretical perspective focuses on the width-scaling relationships between these elements and will omit more specific optimizer features like momentum and gradient or update clipping. SGD represents optimizers where the scale of the update matches the scale of the learning rate times the scale of the gradients. Adam represents adaptive optimizers where, due to the normalization by the gradient scale, the scale of the update matches the scale of the learning rate regardless of the gradient. Finally, Adafactor represents adaptive optimizers with parameter scaling that normalize the gradient similarly to Adam but then multiply by the parameter scale. This results in an update scale that matches the learning rate scale $\times$ the parameter scale; under constant learning rates, the Adafactor updates match the RMS (or Frobenius) norm of the parameters. Note that parameter scaling is the key distinction between the last two regimes: Adam plus parameter scaling falls into the Adafactor family; Adafactor with parameter scaling removed falls into the Adam family.

2.2. Stability, nontriviality and feature learning

Yang & Hu (2021); Yang & Littwin (2023) derive a system of linear constraints on the exponents of the parameterization from the two following concepts: *stability*, where

the activations are exactly constant scale and the logits are no more than constant scale, and *nontriviality*, where the change in logits after the initialization is at least constant scale. Note these constraints are defined solely in terms of the activations and logits, so that only the forward pass is directly constrained and any constraints on backward pass quantities like gradients or updates are indirect consequences. In addition, they define feature learning as a constant scale change after initialization in the activations directly before the readout layer. They also present a full characterization of the infinite-width limits of the space of width-scaling parameterizations as a dichotomy between a feature learning regime and a kernel regime, where feature learning is defined as a constant scale change after initialization in the activations directly before the readout layer.

2.3. Hyperparameter Transfer

Parameterizations with well-understood scaling behavior can enable hyperparameter transfer across scales as relatively cheap hyperparameter search using small models can be used to select hyperparameters for larger, more expensive models (Yang et al., 2022). This transfer is enabled by the particular functional form for each parameterized quantity: the scaling dimension is raised to a theoretically motivated exponent, with a constant multiplicative factor that is determined empirically. When theory holds and the scaling is perfectly encapsulated by the exponent, the optimal empirical constant is the same across scales and can be determined via hyperparameter search on small models and then reused on large models where hyperparameter search would be expensive or impossible. Yang et al. (2022) shows how per-layer learning rates as in muP can be implemented in Transformers, and demonstrates the benefits of muP to hyperparameter transfer across width.

2.4. Alignment

As we will see in Section 3, the conditions for stability differ between the first and subsequent forward passes because of the learning process itself. While the initial random parameters are independent from the data distribution, correlations can develop over time because the updates carry information about the data. Such correlations cause "alignment" between the parameters and activations, in the sense that the vectors may point in similar directions. As such, the norm of the activations after a given layer depends on three quantities: the scale of the input to the layer, the scale of the parameters in the layer, and the alignment between the parameters and the input or "data". In a matrix multiplication, when we sum over the interior dimension n, this alignment contributes a scaling term that is $O(\sqrt{n})$ when there is no alignment and $O(n)$ when there is significant alignment.

The intuition for this calculation can be seen by considering the simpler case of the scaling of the inner product of two random vectors, because the entries in the product of a matrix multiplication are each vector-vector inner products themselves. As the length of the two random vectors becomes large, by straightforward application of the Central Limit Theorem the inner product is a sample from a normal distribution, so its norm has two terms coming from the mean and the variance of this distribution. The takeaway is that the mean term contributes an $O(n)$ term to the norm of the inner product and the variance term contributes $O(\sqrt{n})$. When the mean term is zero because the vectors are zero-mean and independent, then the variance term dominates and the inner product scales like $O(\sqrt{n})$. However, when the vectors are *correlated* or in the worst case are identical, then the inner product scales like $O(n)$ because the coefficient to the mean term is a constant. Yang & Hu (2021) refers to this idea as "Central Limit Scaling" versus "Law of Large Numbers" scaling owing to the idea that the Law of Large Numbers governs how the mean converges and the Central Limit Theorem governs how the variance converges.

This scaling affects the norm of the activations following a layer that multiplies its parameter matrix by the input to that layer. On the first forward pass, due to the random initialization the parameters cannot be aligned with the data, so the $O(\sqrt{n})$ scaling holds. However, during the first backwards pass, the updates to the parameters are a function of the first batch of data, so the parameters may become correlated to the data distribution. As a result, during subsequent forward passes, we can no longer assume perfect independence between the parameters and data and instead the activations might scale with up to an $O(n)$ term. The consequence of this difference in scaling is that the learning rate needs to *counteract* this $O(\sqrt{n})$ or $O(n)$ term, so the maximal stable learning rate is *smaller* by a factor of $O(\sqrt{n})$ when the parameters and data are aligned than when they are not.

3. Theoretical Contributions

In this section we make four theoretical contributions. First, we define a general space of width-scaling parameterizations that explicitly quantifies the contribution of three alignment terms. Rather than making specific assumptions about alignment and then deriving which parameterizations are stable and nontrivial under those assumptions, as in Yang & Hu (2021); Yang & Littwin (2023), we propose general stability and nontriviality constraints as a function of those alignment variables. Second, we propose theory for Adafactor or other adaptive optimizers using parameter scaling. Third, for all parameterizations $\times$ optimizers, we find the maximum stable learning rate for each layer as a function of the alignment terms, and compute the learning rate exponents under two specific alignment assumptions, which we refer to as "full alignment" and "no alignment". While the alignment assumptions in Yang & Hu (2021) prevent standard and NTK

parameterizations from feature learning regardless of the per-layer learning rate prescription, under our assumptions of either full alignment or no alignment all parameterizations have per-layer learning rates in the feature learning limit. Fourth, we propose the *alignment ratio* metric that we will use for empirical investigation, which measures the contribution of alignment to the activations during training.

3.1. Model and Notation

Following a similar model and notation as Yang & Hu (2021), we consider a multilayer perceptron with L hidden layers with weight matrices $W_1 \in \mathbb{R}^{n\times d}$, $W_2, \dots W_L \in \mathbb{R}^{n\times n}$, and $W_{L+1} \in \mathbb{R}^{d\times n}$ and nonlinearity $\phi : \mathbb{R} \to \mathbb{R}$. The parameterization for each layer l is specified by three values $\{a_l, b_l, c_l\}$, where the parameter multiplier is n^{-a_l}, the parameters after t backward passes are W_l^t and are initialized as $W_l^0 \sim \mathcal{N}(0, n^{-2b_l})$, and the learning rate $\eta_l \propto n^{-c_l}$.

For an input $x \in \mathbb{R}^d$, the model that outputs logits z_{L+1} is

$$\begin{aligned} z_1 &= \phi(n^{-a_1} W_1 \cdot x) \\ z_l &= \phi(n^{-a_l} W_l \cdot z_{l-1}) \\ z_{L+1} &= n^{-a_{L+1}} W_{L+1} \cdot z_L \end{aligned}$$

We assume the nonlinearity ϕ has bounded derivative, and that the number of layers L and number of training steps are $O(1)$. These parameterizations occupy equivalence classes under infinite precision owing to our ability to "factor out" a constant term from the initialization into the parameter multiplier (see §B.2). In particular, standard + NTK parameterizations and muP + mean-field parameterization are equivalent under the right learning rates. As neural networks regularly encounter finite-precision effects, we consider all four parameterizations separately. See §B for details.

3.2. Alignment-General Space of Parameterizations

We now propose a general space of parameterizations where we define three alignment variables and define the space of parameterizations that are stable and nontrivial as a function of these variables. Throughout this section, we are interested in the "scale" of quantities, or the exponents of quantities with respect to width in the large-width limit, defined formally in §B.3. A parameterization is *stable* if the activations have exactly constant scale and the logits have at most constant scale, and a parameterization is *nontrivial* if the change in logits after initialization has at least constant scale. To define the alignment variables, we assign α_l, ω_l and u_l to be the exponents of the alignment contributions from three terms $\Delta W_l z_{l-1}, W_l \Delta z_{l-1}$ and $\Delta W_l \Delta z_{l-1}$, respectively, that appear in the expansion $(W_l + \Delta W_l)(z_{l-1} + \Delta z_{l-1}) = (W_l z_{l-1} + \Delta W_l z_{l-1} + W_l \Delta z_{l-1} + \Delta W_l \Delta z_{l-1})$ when computing the activations during training. See §B for formal definitions and a full derivation.

Stability at initialization During the first forward pass, the activations have no alignment because of the random initialization, inducing these constraints regardless of optimizer:

$$\begin{aligned} a_1 + b_1 &= 0 \\ a_l + b_l &= 1/2, \quad l \in [2, \dots, L] \\ a_{L+1} + b_{L+1} &\geq 1/2 \end{aligned}$$

Stability during training We first define r_l as the negative exponent of Δz_l for all l. Then, the constraints $r_l \geq 0$ for $l \in [1, L]$ ensure the activations remain constant scale during training, and the constraint $r_{L+1} \geq 0$ ensures the logits remain at most constant scale during training. In addition, feature learning corresponds to $r_L = 0$.

SGD:
$$\begin{aligned} g_l &:= \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1}) + a_l \\ r_1 &= g_1 + a_1 + c_1 \\ r_l &= \min \begin{cases} g_l + a_l + c_l - \alpha_l \\ g_l + a_l + c_l + r_{l-1} - u_l \\ 1/2 + r_{l-1} - \omega_l \end{cases} \\ r_{L+1} &= \min \begin{cases} 2a_{L+1} + c_{L+1} - \alpha_{L+1} \\ 2a_{L+1} + c_{L+1} + r_L - u_{L+1} \\ a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \end{cases} \end{aligned}$$

Adam:
$$\begin{aligned} r_1 &= a_1 + c_1 \\ r_l &= \min \begin{cases} a_l + c_l - \alpha_l \\ a_l + c_l + r_{l-1} - u_l \\ 1/2 + r_{l-1} + \omega_l \end{cases} \\ r_{L+1} &= \min \begin{cases} a_{L+1} + c_{L+1} - \alpha_{L+1} \\ a_{L+1} + c_{L+1} + r_L - u_{L+1} \\ a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \end{cases} \end{aligned}$$

Adafactor:
$$\begin{aligned} r_1 &= c_1 \\ r_l &= \min \begin{cases} 1/2 + c_l - \alpha_l \\ 1/2 + c_l + r_{l-1} - u_l \\ 1/2 + r_{l-1} + \omega_l \end{cases} \\ r_{L+1} &= \min \begin{cases} a_{L+1} + b_{L+1} + c_{L+1} - \alpha_{L+1} \\ a_{L+1} + b_{L+1} + c_{L+1} + r_L - u_{L+1} \\ a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \\ c_{L+1} \end{cases} \end{aligned}$$

Tensor Programs as a Special Case We recover the stability and nontriviality constraints in Yang & Hu (2021); Yang & Littwin (2023) exactly if and only if $\alpha_l = 1\ \forall l \in [2, L+1]$, $\omega_l = 1/2\ \forall l \in [2, L]$ and $\omega_{L+1} = 1$. This assumption that $\omega_{L+1} = 1$ is at the very core of the theoretical motivation for muP. Feature learning corresponds to $r_L = 0$, and standard and NTK parameterizations both have $a_{L+1} + b_{L+1} = 1/2$. Therefore, due to the logit constraint that $a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \geq 0$, when $\omega_{L+1} = 1$ it is not possible for STP and NTK to achieve feature learning where $r_L = 0$ *regardless of what learning rate prescription is used*. In contrast, muP and MFP shift the readout layer by $\sqrt{n}$ so that $a_{L+1} + b_{L+1} = 1$, so they can attain feature learning where $r_L = 0$ when $\omega_{L+1} = 1$.

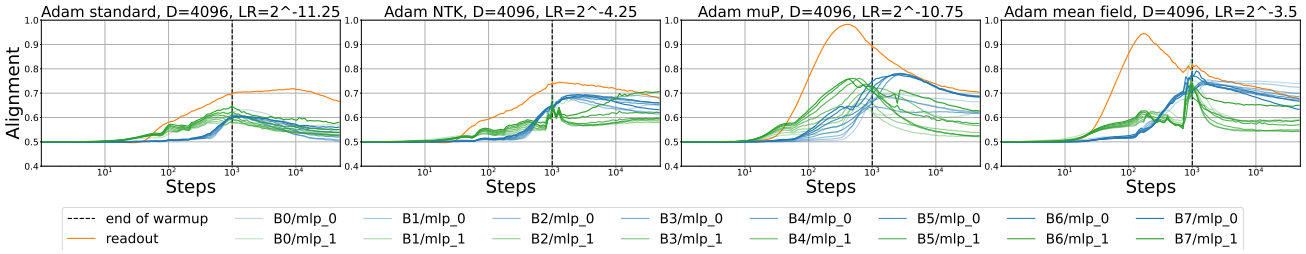

Figure 1. The log alignment ratio metric in readout and hidden (MLP) layers across training steps for each parameterization, for Adam 1.9B parameter models ($H = 32, D = 4096$) using optimal global learning rates.

Maximum Stable Learning Rate Exponents In Table 1 we compute the maximum stable per-layer learning rate exponents under two specific alignment assumptions, giving one prescription for "full alignment" where $\alpha_l = u_l = 1$, and one for "no alignment" where $\alpha_l = u_l = 1/2$ for $l \in [2, L+1]$. In both settings, we assume $\omega_l = 1/2$ for $l \in [2, L+1]$, which in particular relaxes the ω_{L+1} assumption to $1/2$ instead of 1. As a result, our per-layer learning rate prescriptions for standard and NTK are in feature learning limits under our alignment assumptions, a significant difference from prior work.

3.3. Alignment Ratio

In the forward pass, the activations following layer l are computed as $z_l = n^{-a_l}(W_l + \Delta W_l)(z_{l-1} + \Delta z_{l-1})$. The alignment variables $\alpha_l, \omega_l,$ and u_l quantify the alignment in each of the last three terms in the expanded sum $(W_l + \Delta W_l)(z_{l-1} + \Delta z_{l-1}) = W_l z_{l-1} + \Delta W_l z_{l-1} + W_l \Delta z_{l-1} + \Delta W_l \Delta z_{l-1}$. However, we note that in practice the alignment in the terms in this sum may interfere constructively or destructively, and the single alignment quantity that actually governs the scale of the activations z_l is the alignment between $(W_l + \Delta W_l)$ and $(z_{l-1} + \Delta z_{l-1})$.

We therefore propose a metric that measures this alignment, between $(W_l + \Delta W_l)$ and $(z_{l-1} + \Delta z_{l-1})$, in order to understand empirically how alignment that accumulates during training is contributing to the activation scales throughout the model. This metric is defined on each dense layer and quantifies the contribution from alignment between the current parameters and the current pre-layer activations on the scaling exponent of the post-layer activations.

Consider a neural network layer l at training step t with parameters $W_l^t \in \mathbb{R}^{n \times m}$, and pre-layer activations $z_{l-1}^t \in \mathbb{R}^{B \times n}$, where B is the batch size, n is the fan-in and m is the fan out. The matrix multiplication between the pre-layer activations and parameters has an interior dimension n, so this is the base of the exponent to the alignment term.

Definition 3.1. We define the *log alignment ratio* as

$$A_l^t = \log_n \frac{\|z_{l-1}^t W_l^t\|}{\|z_{l-1}^t\| \|W_l^t\|} \in \mathbb{R},$$

where the norm is the root-mean-square (RMS) norm.

4. Experiments

We investigate the role of alignment, per-layer learning rate exponents and constant factors, and the epsilon hyperparameter by running *tens of thousands* of experiments in a decoder-only Transformer language model (Liu et al., 2024) across all combinations of the three optimizers, four parameterizations, learning rate sweeps with a granularity of $2^{0.25}$ or $2^{0.5}$, and fourteen model widths ranging up to $26.8B$ parameters. We include additional experiment results that suggest that our conclusions should hold when using small amounts of weight decay (§F), that Adam + parameter scaling and Adafactor have very similar behavior and occupy the same width-scaling regime (§G), and that moving from the fixed step setting to the compute optimal setting likely requires sharper decay in the learning rate exponents (§H). See §C for more experiment details.

4.1. Alignment Experiments

We measure the log alignment ratio throughout training for three model sizes for each parameterization $\times$ optimizer, using a global learning rate that is close to optimal. For Adam, alignment values for the readout and MLP layers are shown for model dimension $D = 4096$ in Figure 1 and full results including the other optimizers and the dense layers within the attention block (query, key, value, and output projection dense layers) are included in Appendix C.3. As expected, the measured alignment values start at 0.5 due to the independence between the random initialization and the data, and change during training as parameters become aligned with the data or parameters in earlier layers.

The alignment values vary significantly across the training horizon and the trajectories depend heavily on the parameterization and layer type. We see similar values across three model sizes, suggesting that our measurements are likely indicative of the large width behavior. For SGD, the results show high instability and are difficult to interpret; one consistent pattern is that NTK and MFP have almost no alignment and STP and muP have low amounts. Adam and Adafactor show matching trends: the readout layer has the highest peak among the layers, with high peak readout alignment above 0.9 in muP and mean-field parameteriza-

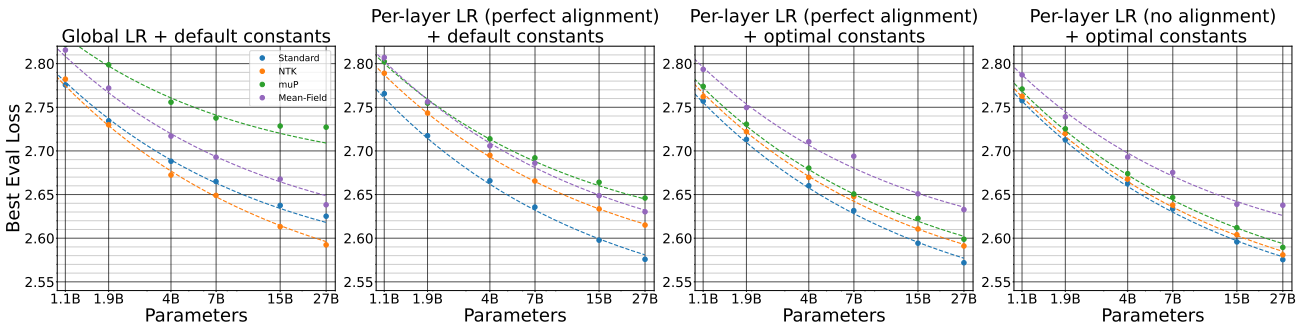

Figure 2. For Adam, eval loss comparisons for all parameterizations across a sequence of interventions. From left to right panels: (a) global lr exponents + default constants, (b) per-layer lr exponents assuming perfect alignment + default constants, (c) per-layer lr exponents assuming perfect alignment + optimal constants, (d) per-layer lr exponents assuming no alignment + optimal constants.

tion and only moderate peak readout alignment between 0.7 and 0.8 in standard and NTK parameterizations. In the MLP layers, alignment in all parameterizations is moderate and does not exceed 0.8.

Overall, these results indicate that the alignment assumptions in Yang & Hu (2021); Yang & Littwin (2023) may be overly conservative; if in practice alignment contributes less than one to the activation exponents, then the learning rate exponents can be larger. However, even given these measurements for alignment, it is not obvious how exactly the learning rate exponents that control the base or peak learning rate should be adjusted. First, we see that alignment is a dynamical quantity that varies widely throughout training: even when the readout alignment is close to maximum early in training for muP and MFP, it is much lower for the vast majority of training steps. In addition, alignment varies across layers within the same layer type, which typically use the same learning rate or at least the same learning rate exponent. Further, the interaction between the alignment measurement and learning rate schedule is complex. The learning rate schedule likely influences the alignment measurements, and alignment likely influences the optimal learning rate schedule: one possible role of learning rate schedules is that the decay counteracts alignment that develops later in training. Even if we used alignment measurements from experiments under one set of learning rates to inform adjustments to the learning rate exponents, this would induce an iterative loop where the adjusted learning rates would then affect the alignment. We will therefore take an empirical approach to determine how alignment should influence the learning rate exponents, and consider several choices of alignment assumptions for the per-layer learning rate experiments in the following section.

4.2. Per-layer Learning Rates

As in Table 1, all parameterizations have per-layer learning rate exponents specific to the optimizer and the choice of alignment assumption. In this section, we empirically validate these theoretical learning rate exponent prescriptions and investigate the impact of tuning the per-layer constant factors. We compare against global learning rates as a baseline: while global learning rates are in most cases not theoretically principled, they are the overwhelmingly dominant paradigm in practice. In most settings, the theoretical learning rate exponents differ across layers, implying a mismatch in at least some layers when using global learning rates. However, there are certain cases, such as muP + SGD + full alignment, or Adafactor + any parameterization + no alignment, where the theoretically motivated per-layer exponents happen to be the same in all layers so that global learning rate actually coincides with the theoretical prescription. We include experiments using the theoretical prescriptions from both the full alignment and no alignment settings since our empirical alignment measurements show intermediate and highly dynamic values.

Our first experiment compares per-layer learning rates when assuming full alignment against the baseline of global learning rates. For all experiments, we select a base model dim of $b = 1024$ and define the learning rate in layer l as $\eta_l = \beta_n \cdot \frac{n}{b}^{-c_l}$. We determine the best β_n for each model dim n with a one-dimensional sweep and report the eval loss from the best value. For global learning rates, $c_l = 0$ for all l, and for per-layer learning rates, c_l follows Table 1. Since there is no contribution from the learning rate exponents at the base model size, the global and per-layer learning rate settings coincide exactly at the base model dim and differ only when scaling away from the base model size. The per-layer results in Figure 2(b), compared with the global learning rate results in Figure 2(a), show that standard, muP and mean-field parameterizations all improve significantly with per-layer learning rates. The notable exception is that NTK actually performs worse using the prescribed per-layer exponents: we note that these exponents assume full alignment and return to this result later in this section.

We next introduce constant multiplicative factors to the per-layer learning rates and propose a hyperparameter transfer strategy where we tune these constants at small scale and reuse them across model sizes. Again using the base model

dim of $b = 1024$, we now define the learning rate in layer l as $\eta_l = \beta_n \cdot \gamma_l \cdot \frac{n}{b}^{-c_l}$ where γ_l is this constant factor that should be determined empirically. We use the same learning rate for all hidden layers, so we need to determine γ_1 for the embedding layer, γ_h for the hidden layers, and γ_{L+1} for the readout layer. The previous experiments correspond to setting these constants equal to one by default. We continue to sweep one dimension at all model sizes to determine β_n, so we note the choice of $(\gamma_1, \gamma_h, \gamma_{L+1})$ really defines two ratios as a common factor can be absorbed by β_n.

We tune $(\gamma_1, \gamma_h, \gamma_{L+1})$ at the base model dim using a three-dimensional hyperparameter search described in Appendix C and reuse the best set of values across all model sizes. For the base model dim, tuning these constants can only improve performance, but comparing Figure 2(c) to 2(b) shows these constants improve performance substantially for all model sizes across all parameterizations. The only exception is that these constants may not transfer well to the mean-field parameterization models above 2B parameters: we will show in Section 4.3 that this is due to epsilon underflow and is addressed with our epsilon mitigations. This result indicates that this recipe for these per-layer constant multiplicative factors is both essential and practical: the performance gains are substantial and the hyperparameter transfer makes them feasible. If instead, these constants improved performance only at or near the model sizes where they were tuned, it would be impractical to include them for large model training because the three-dimensional sweep would be prohibitively expensive at large scale. While muTransfer (Yang et al., 2022) emphasizes that muP is the unique parameterization that allows hyperparameter transfer across width, this result shows that *all parameterizations can perform hyperparameter transfer* across width when each parameterization uses theoretically motivated per-layer learning rate exponents.

Yang et al. (2022) also presents empirical results in which muP outperforms standard parameterization. We note, however, that this comparison across parameterizations has several elements that favor muP. First, the muP experiments use per-layer learning rate scaling while the standard parameterization experiments use a global learning rate. Second, the muP results tune a handful of constant factors at small scale and transfer them to large scale. While the specific constant factors differ from our setting, this is comparable to using our optimal constant learning rate factors instead of the default constant factors. As such, their comparison of muP and standard parameterization is analogous to comparing muP in our third experiment (Figure 2(c), green curve) against standard parameterization in our first experiment (Figure 2(a), blue curve), which indeed shows a benefit for muP. Instead, we argue that the fair comparison across parameterizations would use per-layer learning rates and optimal constants for both, that is, to consider both parameterizations in our third experiment Figure 2(c). There, we see that standard parameterization outperforms muP and in fact substantially so: the *second largest* standard parameterization model outperforms the *largest* muP model despite having 57% as many parameters.

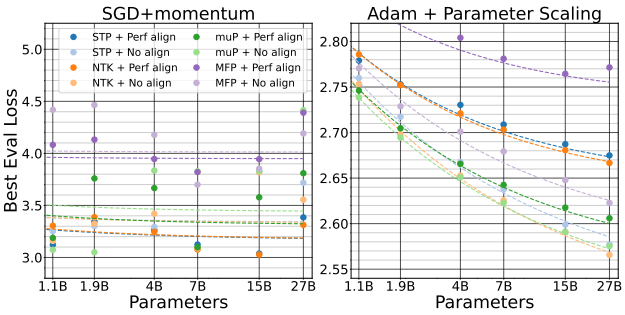

Figure 3. SGD and Adam+parameter scaling for all parameterizations × full and no alignment assumptions + optimal constants.

Finally, we investigate how alignment assumptions impact performance, still using per-layer learning rates and optimal constants. In Figure 2(d), we use the learning rate exponents from the right side of Table 1 derived from assuming no alignment rather full alignment. This yields surprising results given the focus on alignment in prior work: for Adam, all parameterizations except standard parameterization benefit from using the no alignment assumption. In contrast, standard parameterization shows better performance using full alignment where the difference is slight but consistent across scale. Moreover, for Adam with parameter scaling, Figure 3 shows that all parameterizations perform significantly better using no alignment rather than full; the performance gap increases with scale suggesting the no alignment exponents are truly preferable. Finally, the best performing model across all parameterizations and optimizers is quite unexpected: it is Adam + param scaling + NTK + no alignment! This result further validates that it is critical to consider our more general space of parameterizations rather than making specific alignment assumptions upfront.

We note these empirical results could occur due to a mechanism other than the activation or logit growth with respect to width that is predicted by alignment. For example, prior work notes training instabilities in Transformers due to attention logit growth (Dehghani et al., 2023; Zhai et al., 2023) or output logit divergence (Chowdhery et al., 2023) and that these instabilities are sensitive to the learning rate (Wortsman et al., 2023). We may also miss slight undercorrection of the alignment in our finite size models: if the true alignment is slightly larger than the learning rate exponents account for, we could have slow growth of activations or logits with respect to width that does not harm performance in our models but would eventually induce instability at sufficient width. However, we note our largest model with $26.8B$ parameters has a model dimension of $16,384$ which encompasses the width of many even larger models.

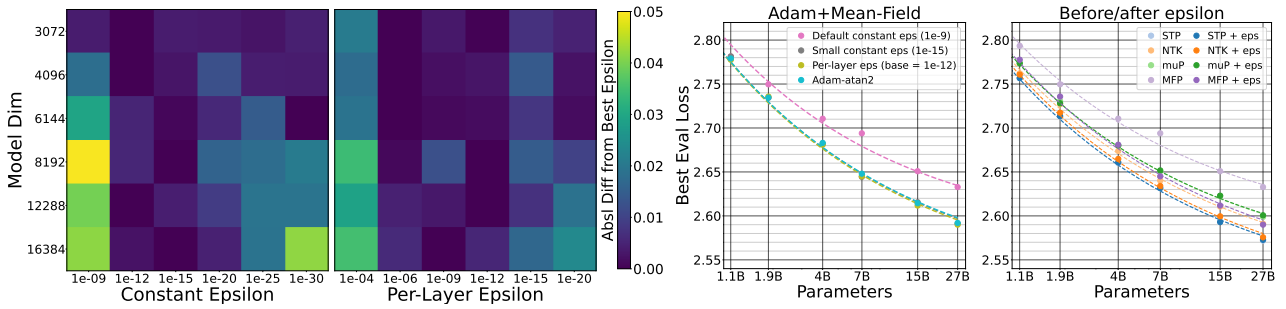

Figure 4. (a-b) Heatmaps for hyperparameter sweeps of epsilon constant factors: color indicates the absolute difference in eval loss from the best constant factor for that setting. Dark blue columns in the middle indicate good performance whereas smaller or larger values have suboptimal performance. (c) Epsilon mitigations for Adam + mean-field using the best choice of constants in each mitigation setting, showing all three mitigations equivalently and substantially improve performance. (d) All parameterizations using default epsilon (dark colors) and per-layer epsilon (light colors) show MFP has large benefit from fixing epsilon and NTK has moderate benefit. In addition, after epsilon mitigation, equivalent parameterizations (STP+NTK, muP+MFP) have approximately equivalent performance.

4.3. Epsilon Underflow in Adaptive Optimizers

In adaptive optimizers like Adam, the denominator of the update rule adds a small epsilon parameter to the gradient second-order moment, originally intended to regularize against division by zero when the gradients are very small (Duchi et al., 2011; Hinton et al., 2012). Despite its small value, typically around 1e-8 (Paszke et al., 2019; Babuschkin et al., 2020), the epsilon parameter prevents Adam from being perfectly scale-invariant where multiplying the gradients by a constant would not alter the resulting update. In particular, if the gradient scale drops below the size of epsilon then epsilon dominates the gradients instead of acting as a negligible additive constant. Since gradients decrease in scale with model width as in Table 1, for any constant value of epsilon there exists a sufficiently large model that will encounter this scenario.

Two recent works note this phenomenon and propose possible mitigations. From an empirical perspective, Wortsman et al. (2023) observes that gradient norms in standard parameterization models decrease with model size and approach 1e-8 for 1.2B parameter models, suggesting this epsilon underflow is relevant in practice. As a mitigation, they propose using a smaller constant value for epsilon and show that decreasing epsilon from 1e-8 to 1e-15 improves the loss in a 4.6B parameter model. From a theoretical perspective, Yang & Littwin (2023) notes that epsilon should be treated as part of the parameterization and propose per-layer epsilon scaling. For each layer, epsilon is proportional to the parameterization and layer-specific gradient scale shown in Table 1. Similar to our per-layer learning rate experiments, we implement this as $\epsilon_l = \text{base epsilon} \cdot (\frac{n}{b})^{-g_l}$ for each layer l, where g_l is the negative exponent of the gradient scale, the base model dim b is 1024, and the base epsilon is determined empirically. This approach has not been empirically validated and adds significant implementation complexity, but should ensure that epsilon and the gradients scale in tandem across width.

We investigate the practical impact of epsilon across parameterizations. As seen in the gradients column of Table 1, gradients decrease as model width increases with an exponent specific to both the parameterization and layer. As such, we expect that different parameterizations will encounter epsilon underflow at different model sizes: in particular, the steepest exponent is the mean-field parameterization hidden layer, which scales like $1/D^{1.5}$. This suggests that mean-field parameterization should encounter epsilon underflow at smaller model sizes than other parameterizations.

Both constant epsilon and per-layer epsilon require hyperparameter tuning, to select the constant or the base constant multiplier, respectively. As shown in Figure 4(a-b), values that are too large lead to suboptimal performance; values that are too small lead to instability, presumably by failing to prevent the numerical instability epsilon was originally intended to prevent. Rather than providing numerical stability for small number division with an additive constant in the denominator that breaks scale-invariance, we propose *Adam-atan2*: a variant of Adam that replaces the standard division operation in the update rule with the standard library function arctangent, which performs numerically stable small number division and is defined even at $(0, 0)$ exactly. The single-line code change in Appendix C.3 to use arctangent in the Adam update equation eliminates the epsilon hyperparameter entirely and restores the scale invariance to Adam.

For all parameterizations, we compare the three mitigations using the best choice of constant in each setting against a baseline epsilon (1e-9): small constant epsilon (1e-15), per-layer epsilon scaling (base = 1e-12), and our proposal *Adam-atan2*. In Figure 4(c), we see that all three mitigations have noticeable improvements in the eval loss in mean-field parameterization and that the gap increases with model size. The other parameterizations, seen here in Figure 4(d) and in Figure 8 in the appendix, show similar but more subtle improvements for NTK and no improvements for standard and muP. The particular sensitivity of mean-field

parameterization to epsilon is consistent with our theoretical motivation: due to its hidden layer gradients scaling like $1/D^{1.5}$, we expect that among the parameterizations, mean-field will encounter epsilon underflow at the smallest model sizes and benefit most from these mitigations.

We therefore recommend care when setting epsilon. For standard parameterization models with up to a billion parameters, the typical default value of 1e-8 is likely acceptable but slightly smaller values of 1e-12 or 1e-15 may be preferable. For larger models or other parameterizations, using epsilon requires smaller constants or per-layer epsilon scaling, with tradeoffs between implementation complexity and, at least theoretically, hyperparameter tuning costs. In principle, the theoretical prescription for per-layer epsilon encapsulates the epsilon scaling in the exponents, allowing hyperparameter transfer of the constant multiplier similar to other parameterized quantities like learning rates. In contrast, the optimal constant epsilon should be scale-dependent in theory, but we note that with our model sizes and constant search resolution we did not see scale dependence of the optimal epsilon constant. However, this epsilon hyperparameter can be eliminated entirely with Adam-atan2 with a one-line code change and the same improved performance.

Lastly, epsilon illustrates that finite precision plays an important role in parameterization in practice. Recall that standard and NTK parameterizations, and similarly muP and mean-field parameterizations, are theoretically equivalent under the equivalence relations in Appendix B.2. Yet, before addressing epsilon we see significant performance gaps between equivalent parameterizations, which are closed when epsilon underflow is mitigated: now the pairs standard + NTK and muP + mean-field show approximately equal performance. Moreover, this illustrates that the standard parameterization equivalence class outperforms the entire muP equivalence class, which narrows the possible explanations for the performance differences to the readout layer, as this is the primary difference between the equivalence classes.

5. Related Work

Prior work on width-scaling parameterizations (Lee et al., 2017; Matthews et al., 2018; Jacot et al., 2018) includes the neural tangent kernel parameterization (Jacot et al., 2018), a modification to standard parameterization to enable a consistent infinite-width limit (Sohl-Dickstein et al., 2020), and a mean-field limit (Mei et al., 2018; Geiger et al., 2020; Rotskoff & Vanden-Eijnden, 2018; Chizat & Bach, 2018; Araújo et al., 2019) of single-hidden-layer MLPs. Bordelon & Pehlevan (2022) proposed the mean-field parameterization by extending this mean-field limit to deep neural networks using self-consistent dynamical field theory. Yaida (2022) proposed a one-parameter family of hyperparameter scaling strategies that interpolates between the neural tangent scaling and mean-field or muP scaling. This meta-parameterized scaling strategy has been used to propose width-scaling initialization and training hyperparameters in Transformers (Dinan et al., 2023). Various "unit scaling" strategies proposed by Shazeer (2020); Kaplan (2019) are similar to NTK parameterization; in addition Blake et al. (2023) also modifies the per-layer gradients to keep the activations and parameters close to unit scale regardless of model size. Yang et al. (2023) proposes a spectral perspective on muP and Ishikawa & Karakida (2023) extend muP to second-order optimizers (K-FAC, Shampoo).

6. Conclusions and Future Work

From our broad perspective across parameterizations and optimizers, we find that key assumptions about alignment in prior work require additional consideration. Using empirical measurements from our alignment ratio metric, we find that alignment is a dynamical quantity that depends significantly on the training step, parameterization and layer, and less heavily on the optimizer. The alignment measured during training gives intermediate values that indicate that alignment assumptions in prior work may be overly conservative, suggesting that a larger set of parameterizations is more interesting than previously thought.

By considering a more general space of parameterizations with respect to the alignment, we show that all parameterizations benefit from theoretically motivated learning rate exponent prescriptions. We also demonstrate that several hyperparameters should be chosen carefully. First, we show the necessity and practicality of tuning the constant factors in per-layer learning rate prescriptions. These constants transfer well across model sizes, showing that all parameterizations can perform hyperparameter transfer under the right theoretical prescription. Second, the epsilon hyperparameter in adaptive optimizers induces gradient underflow using typical defaults at realistic model sizes, in particular for mean-field parameterization. Theoretically, epsilon should be considered part of the parameterization and scaled per-layer, but practically, small constant values can perform just as well when selected carefully. To eliminate epsilon entirely, we propose making Adam scale-invariant with *Adam-atan2*.

Future work might consider alignment-aware learning rate schedules or alignment-aware optimizers. In addition, since the characterization of parameterizations into feature learning and kernel limits is specific to the alignment assumptions, this characterization could be extended to the general alignment setting. Beyond width scaling, future work should investigate the other scaling dimensions that are necessary for large model training, in particular depth and batch size.

Acknowledgements

The authors would like to thank Yasaman Bahri and Kevin Yin for detailed feedback on paper drafts, and Kevin Yin for the suggestion to scale the arctangent function as $\lambda \cdot \text{atan2}(x, \lambda y)$ which improved performance. We also thank Ben Adlam, Kelvin Xu, Justin Gilmer, Gamaleldin Elsayed, Mark Kurzeja, Jiri Hron, Noah Fiedel, Zachary Nado, Justin Austin, Ben Poole, Clare Lyle and Tomás Lozano-Pérez for technical discussions and grateful to four anonymous ICML reviewers who provided especially high-quality reviews and discussion during the rebuttal period.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, in particular due to the large model sizes considered in this work, but we do not feel there are specific aspects of this work with broader impacts beyond the considerations relevant to all large machine learning models.

References

Araújo, D., Oliveira, R. I., and Yukimura, D. A mean-field limit for certain deep neural networks. *arXiv preprint arXiv:1906.00193*, 2019.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Papamakarios, G., Quan, J., Ring, R., Ruiz, F., Sanchez, A., Sartran, L., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stanojević, M., Stokowiec, W., Wang, L., Zhou, G., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Blake, C., Orr, D., and Luschi, C. Unit scaling: Out-of-the-box low-precision training. *arXiv preprint arXiv:2303.11257*, 2023.

Bordelon, B. and Pehlevan, C. Self-consistent dynamical field theory of kernel evolution in wide neural networks. *Advances in Neural Information Processing Systems*, 35: 32240–32256, 2022.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chizat, L. and Bach, F. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems*, 31, 2018.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

Dinan, E., Yaida, S., and Zhang, S. Effective theory of transformers at initialization. *arXiv preprint arXiv:2304.02034*, 2023.

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

Geiger, M., Spigler, S., Jacot, A., and Wyart, M. Disentangling feature and lazy training in deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(11):113301, 2020.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 1487–1495. ACM, 2017. doi: 10.1145/3097983.3098043. URL https://doi.org/10.1145/3097983.3098043.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

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

Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Ishikawa, S. and Karakida, R. On the parameterization of second-order optimization effective towards the infinite width. *arXiv preprint arXiv:2312.12226*, 2023.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Kaplan, J. Notes on contemporary machine learning for physicists, 2019.

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

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Liu, P. J., Novak, R., Lee, J., Wortsman, M., Xiao, L., Everett, K., Alemi, A. A., Kurzeja, M., Marcenac, P., Gur, I., Kornblith, S., Xu, K., Elsayed, G., Fischer, I., Pennington, J., Adlam, B., and Dickstein, J.-S. Nanodo: A minimal transformer decoder-only language model implementation in JAX., 2024. URL http://github.com/google-deepmind/nanodo.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

Mei, S., Montanari, A., and Nguyen, P.-M. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33): E7665–E7671, 2018.

Neal, R. M. Priors for infinite networks. *Bayesian learning for neural networks*, pp. 29–53, 1996.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Rotskoff, G. and Vanden-Eijnden, E. Parameters as interacting particles: long time convergence and asymptotic error scaling of neural networks. *Advances in neural information processing systems*, 31, 2018.

Shazeer, N. Mesh TensorFlow - Model Parallelism Made Easier; comments on unit scaling convention, 12 2020. URL https://github.com/tensorflow/mesh/blob/master/mesh_tensorflow/layers.py#L48-L98.

Shazeer, N. and Stern, M. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

Sohl-Dickstein, J., Novak, R., Schoenholz, S. S., and Lee, J. On the infinite width limit of neural networks with a standard parameterization. *arXiv preprint arXiv:2001.07301*, 2020.

Wortsman, M., Liu, P. J., Xiao, L., Everett, K., Alemi, A., Adlam, B., Co-Reyes, J. D., Gur, I., Kumar, A., Novak, R., et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020.

Yaida, S. Meta-principled family of hyperparameter scaling strategies. *arXiv preprint arXiv:2210.04909*, 2022.

Yang, G. and Hu, E. J. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Yang, G. and Littwin, E. Tensor programs ivb: Adaptive optimization in the infinite-width limit. *arXiv preprint arXiv:2308.01814*, 2023.

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., and Gao, J. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

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

Zhai, S., Likhomanenko, T., Littwin, E., Busbridge, D., Ramapuram, J., Zhang, Y., Gu, J., and Susskind, J. M. Stabilizing transformer training by preventing attention entropy collapse. In *International Conference on Machine Learning*, pp. 40770–40803. PMLR, 2023.

Appendix

A. Author Contributions

Katie, Lechao, Jaehoon and Jeffrey were the four core project contributors. All core contributors were closely involved throughout the duration of the project and made contributions to developing the theory, analyzing and debugging experiments, framing the narrative, reviewing code and giving feedback on writing. Katie led the project and led the theory, implemented and ran all experiments, produced all figures, and wrote the paper. Lechao made particular contributions to theory. Jaehoon made particular contributions to experimental design and framing the paper in relation to prior work, including drafting the related work section. Jeffrey made particular contributions to theory, experiment analysis, and paper framing and provided the primary advising on the project.

The experiments were implemented on top of a base Transformer model codebase. Peter led the development of this base model codebase and Roman, Mitchell, Jaehoon, Lechao and Katie made significant contributions to this codebase.

In addition, Mitchell contributed expertise on optimizers and weight decay. Alex contributed expertise on quantifying the uncertainty in exponent measurements. Roman implemented a small library used for reparameterization and gradient scaling. Jascha contributed to early discussions on parameterization and gradient scaling, contributed ideas about weight decay, and gave feedback on writing. Izzeddin contributed to technical discussions on alignment. Leslie contributed to technical discussions and gave feedback on paper framing.

B. Theoretical Details

In this section, we provide formal definitions and a complete derivation of the constraints for our alignment-general space of parameterizations.

B.1. Model

Following a similar model and notation as Yang & Hu (2021), we consider a multilayer perceptron with L hidden layers with weight matrices $W_1 \in \mathbb{R}^{n\times d}$, $W_2, \dots W_L \in \mathbb{R}^{n\times n}$, and $W_{L+1} \in \mathbb{R}^{d\times n}$ and nonlinearity $\phi : \mathbb{R} \to \mathbb{R}$. The parameterization for each layer l is specified by three quantities $\{a_l, b_l, c_l\}$ which are the negative exponents for the parameter multiplier, initialization standard deviation, and learning rate, respectively. More precisely, for layer l the parameter multiplier is n^{-a_l}, the parameters after t backward passes are W_l^t and are initialized as $W_l^0 \sim \mathcal{N}(0, n^{-2b_l})$, and the learning rate η_l is proportional to n^{-c_l} with a width-independent proportionality constant.

that is determined empirically. For our derivations we will omit that constant and write $\eta_l = n^{-c_l}$.

For an input $x \in \mathbb{R}^d$, we have

$$\begin{aligned} z_1 &= \phi(n^{-a_1} W_1 \cdot x) \\ z_l &= \phi(n^{-a_l} W_l \cdot z_{l-1}) \\ z_{L+1} &= n^{-a_{L+1}} W_{L+1} \cdot z_L \end{aligned}$$

where $z_{L+1}(x)$ gives the logits from the model.

There can be additional values prescribed in a width-scaling parameterization beyond the initialization scale, parameter multipliers and learning rate. For example, Yang & Littwin (2023) includes the epsilon hyperparameter, gradient clipping and weight decay in the parameterization for adaptive optimizers like Adam.

B.2. Equivalence classes

The parameterizations occupy equivalence classes owing to our ability to "factor out" a constant term from the parameter initialization into the parameter multiplier, which exactly preserves the output of the forward pass while multiplying the gradients by this constant. (Yang & Hu, 2021) This change in the gradients can then be "corrected for" by modifying the learning rate in an optimizer-specific manner. We omit any contribution from the epsilon parameter in the adaptive optimizers, equivalent to assuming that epsilon is zero.

In this one-dimensional symmetry group parameterized by θ, to preserve the forward pass, regardless of optimizer,

$$a_l \leftarrow a_l + \theta$$
$$b_l \leftarrow b_l - \theta$$

Then specific to the optimizer, to preserve the effect of the backwards pass, correct the learning rate according to

$$\text{SGD: } c_l \leftarrow c_l - 2\theta$$
$$\text{Adam: } c_l \leftarrow c_l - \theta$$
$$\text{Adafactor: } c_l \leftarrow c_l$$

These equivalence classes were presented for SGD and Adam in Yang & Hu (2021) and Yang & Littwin (2023) respectively, and we propose this relation for Adafactor.

B.3. Defining "scale"

We are interested in the "scale" of various quantities in the infinite-width limit, specifically the exponent with respect to width n as the width becomes large.

Definition B.1. We say that the *scale* of a quantity U is n^v if

$$v = \log_n \lim_{n\to\infty} \|U\|_{RMS}$$

where the norm is the root-mean-square (RMS) norm. Intuitively, the RMS norm describes the size of "typical" entries in a matrix: if all entries were the same then the RMS norm would match the value of each entry.

We use standard Big O notation or the $\sim$ symbol to denote this asymptotic behavior and write:

$$U = \Theta(n^v) \text{ or } U \sim n^v \text{ if } v = \log_n \lim_{n\to\infty} \|U\|_{RMS}$$
$$U = O(n^v) \text{ if } v \geq \log_n \lim_{n\to\infty} \|U\|_{RMS}$$
$$U = \Omega(n^v) \text{ if } v \leq \log_n \lim_{n\to\infty} \|U\|_{RMS}$$

B.4. Assumptions and Notation

We will assume the following:

- The input data is $\Theta(1)$.
- The number of layers L is $O(1)$.
- The number of training steps T is $O(1)$.
- The batch size is one.
- The input and output dimensionality d is $O(1)$.
- The nonlinearity ϕ has bounded (weak) derivative, so the derivative of the nonlinearity does not contribute to the exponent in the infinite-width limit. As such, we omit the nonlinearity in the following calculations, equivalent to assuming ϕ is the identity function.
- We assume that the derivative of the loss with respect to the logits $\nabla_{z_{L+1}}\mathcal{L}$ is $\Theta(1)$. Since the output dimensionality d is $O(1)$, this assumption holds for many common loss functions.
- Our theoretical derivations use real numbers, i.e. assuming infinite precision, despite possible effects from finite precision in practice.
- We denote the difference in a quantity after initialization as $\Delta\bullet^t := \bullet^t - \bullet^0$, for example $\Delta z_l^t := z_l^t - z_l^0$.
- When the timestep is clear from the context, we omit the superscripts.
- Unless otherwise stated, a norm refers to the RMS norm.
- The learning rate η_l is proportional to n^{-c_l} with a width-independent proportionality constant that is typically determined empirically. For our derivations we will omit the proportionality constant and write $\eta_l = n^{-c_l}$.

- We use $\nabla_{W_l}\mathcal{L}$ and $\frac{\partial \mathcal{L}}{\partial W_l}$ interchangeably.

B.5. Defining stability and nontriviality

We will use the definitions from Yang & Hu (2021) for stability and nontriviality:

Definition B.2. A parameterization is *stable* if the activations have exactly constant scale, i.e. $z_l^t = \Theta(1) \ \ \forall l \in [1, L]$ and the logits are at most constant scale, i.e. $z_{L+1} = O(1)$, at all timesteps $0 \leq t \leq T$ during training.

Definition B.3. A parameterization is *nontrivial* if the change in logits after initialization is at least constant scale, i.e. $z_{L+1}^t - z_{L+1}^0 = \Omega(1)$ for some timestep $0 \leq t \leq T$ during training.

The specific choice to require exactly constant scale activations and at most constant scale logits should be thought of as a design choice from which theoretical results follow rather than a theoretical result itself.

B.6. First Forward Pass: Stability at initialization:

The stability constraints at initialization ensure that all intermediate activations z_l are $\Theta(1)$ and the logits z_{L+1} are $O(1)$. The constraints apply iteratively across $O(1)$ layers: since the input x is $O(1)$, the constraint on the first layer ensures that z_1 is $O(1)$, then the constraint on layer l ensures that z_l is $O(1)$ assuming the previous layer $l-1$ constraint are satisfied so that z_{l-1} is $O(1)$.

This gives the constraints for stability at initialization:

$$\begin{aligned} a_1 + b_1 &= 0 \\ a_l + b_l &= 1/2, \quad l \in [2, \ldots, L] \\ a_{L+1} + b_{L+1} &\geq 1/2 \end{aligned}$$

B.7. Gradients at Initialization

At initialization, the gradients for each layer can be calculated using straightforward application of the chain rule. We first define g_l^t as the negative exponent of the gradient scale of the loss with respect to the parameters in layer l at timestep t.

Definition B.4. Let $g_l^t = -\log_n \lim_{n\to\infty} \left\| \frac{\partial \mathcal{L}}{\partial W_l^t} \right\|$ so that $\frac{\partial \mathcal{L}}{\partial W_l} = \Theta(n^{-g_l})$.

Then by the chain rule, the gradient decomposes as

$$\frac{\partial \mathcal{L}}{\partial W_l} = \frac{\partial \mathcal{L}}{\partial z_{L+1}} \frac{\partial z_{L+1}}{\partial z_L} \cdots \frac{\partial z_{l+1}}{\partial z_l} \frac{\partial z_l}{\partial W_l}$$

where $\frac{\partial z_{L+1}}{\partial z_L} = \Theta(1)$ by assumption, $\frac{\partial z_l}{\partial z_{l-1}} = n^{-a_l} \cdot n^{-b_l}$ and $\frac{\partial z_l}{\partial W_l} = n^{-a_l}$.

After taking the logarithm and flipping the negative signs, this gives

$$g_l = a_l + \sum_{i=l+1}^{L} (a_i + b_i - 1/2) + a_{L+1} + b_{L+1}$$

If we then assume the stability at initialization constraints, the terms inside the sum for all hidden layers cancel, leaving that the gradients at initialization are:

$$\begin{aligned} g_{L+1} &= a_{L+1} \\ g_l &= a_l + a_{L+1} + b_{L+1} \text{ for } l \in [1, \ldots, L] \end{aligned}$$

B.8. Optimizer Update Rules

We write out the version of the update rules that we use for these derivations for each optimizer family, which include the aspects that are essential to the scaling exponents but omit more specific features like momentum or moving averages, learning rate schedules, weight decay, clipping, and low-rank factoring. For intuition, it is useful to consider the relationship between the scale of the updates, gradients, parameters, and learning rate for each optimizer. In SGD, the scale of the update matches the scale of the learning rate times the scale of the gradients. In Adam, or similar adaptive optimizers that

normalize by the gradient scale, the scale of the updates match the scale of the learning rate regardless of the gradient scale. In Adafactor, Adam with parameter scaling, or similar optimizers that normalize by the gradient scale and then multiply by the parameter scale, the scale of the updates matches the scale of the learning rate times the scale of the parameters.

$$\begin{aligned}\text{SGD: } \Delta W_l &= \eta_l \cdot \nabla_{W_l} \mathcal{L} \\ \text{Adam: } \Delta W_l &= \eta_l \cdot \frac{\nabla_{W_l} \mathcal{L}}{\|\nabla_{W_l} \mathcal{L}\|} \\ \text{Adafactor: } \Delta W_l &= \eta_l \cdot \|W_l\| \cdot \frac{\nabla_{W_l} \mathcal{L}}{\|\nabla_{W_l} \mathcal{L}\|}\end{aligned}$$

B.9. First Backward Pass

Using the update rules in the previous section, we write out the update for each optimizer during the first backward pass. We note here that so far the calculations and constraints have been the same for all optimizers, and this step is the first one that is specific to the optimizer based on its update rule.

$$\begin{aligned}\text{SGD: } \Delta W_l &= \eta_l \cdot \nabla_{W_l} \mathcal{L} \sim n^{-c_l} \cdot n^{-g_l}, \\ \text{where } g_{L+1} &= a_{L+1} \text{ or } g_l = a_l + a_{L+1} + b_{L+1}, \\ \text{so } \Delta W_l &\sim n^{-a_{L+1}-b_{L+1}-a_l-c_l}, \\ \Delta W_{L+1} &\sim n^{-a_{L+1}-c_{L+1}}\end{aligned}$$

$$\begin{aligned}\text{Adam: } \Delta W_l &= \eta_l \cdot \frac{\nabla_{W_l} \mathcal{L}}{\|\nabla_{W_l} \mathcal{L}\|} \sim n^{-c_l} \cdot 1 \\ \text{so } \Delta W_l &\sim n^{-c_l}\end{aligned}$$

$$\begin{aligned}\text{Adafactor: } \Delta W_l &= \eta_l \cdot \|W_l\| \cdot \frac{\nabla_{W_l} \mathcal{L}}{\|\nabla_{W_l} \mathcal{L}\|} \sim n^{-c_l} \cdot n^{-b_l} \cdot 1 \\ \text{so } \Delta W_l &\sim n^{-c_l-b_l}\end{aligned}$$

B.10. Defining the activation update residual

We will define r_l for all l in $[1, L+1]$ to be the negative exponent of Δz_l. We refer to this as a "residual" quantity because r_l is exactly zero when the updates are maximal such that any increase would cause the activation z_l to exceed constant scale.

Definition B.5. Let $r_l \coloneqq -\log_n \lim_{n\to\infty} \|\Delta z_l\|$, so that $\Delta z_l \sim n^{-r_l}$.

B.11. Defining Alignment Variables

In this section, we will define three alignment variables α_l, ω_l and u_l that are the exponents of the alignment contributions from the $\Delta W_l \cdot z_{l-1}$, $W_l \cdot \Delta z_{l-1}$ and $\Delta W_l \cdot \Delta z_{l-1}$ terms respectively.

Starting in the second forward pass, each activation for layer l in $[2, L+1]$ expands into four terms:

$$\begin{aligned}z_l &= n^{-a_l}(W_l + \Delta W_l)(z_{l-1} + \Delta z_{l-1}) \\ &= n^{-a_l}(W_l \cdot z_{l-1} + \cdot W_l \Delta z_{l-1} + \Delta W_l \cdot z_{l-1} + \Delta W_l \cdot \Delta z_{l-1})\end{aligned}$$

Due to the random initialization, the $W_l \cdot z_{l-1}$ term has no alignment. For the remaining three terms, we introduce the following alignment variables.

Definition B.6. We define the alignment variables as

$$\alpha_l = \log_n \lim_{n\to\infty} \frac{\|\Delta W_l z_{l-1}\|}{\|\Delta W_l\|\|z_{l-1}\|} \quad \text{so that} \quad \Delta W_l z_{l-1} \sim n^{\alpha_l}\|\Delta W_l\|\|z_{l-1}\|,$$
$$\omega_l = \log_n \lim_{n\to\infty} \frac{\|W_l \Delta z_{l-1}\|}{\|W_l\|\|\Delta z_{l-1}\|} \quad \text{so that} \quad W_l \Delta z_{l-1} \sim n^{\omega_l}\|W_l\|\|\Delta z_{l-1}\|,$$
$$u_l = \log_n \lim_{n\to\infty} \frac{\|\Delta W_l \Delta z_{l-1}\|}{\|\Delta W_l\|\|\Delta z_{l-1}\|} \quad \text{so that} \quad \Delta W_l \Delta z_{l-1} \sim n^{u_l}\|\Delta W_l\|\|\Delta z_{l-1}\|.$$

We have omitted the timestep superscripts above, but alignment is a dynamic quantity so more formally we have

$$\alpha_l^t \coloneqq \log_n \lim_{n\to\infty} \frac{\left\|\Delta W_l^t z_{l-1}^t\right\|}{\|\Delta W_l^t\|\left\|z_{l-1}^t\right\|}$$

and similarly for ω_l^t and u_l^t.

Note that for many quantities in our notation, we define the variable to be a negative exponent, but for these alignment variables we are defining α_l, ω_l, u_l as positive exponents so they take on values between 0 and 1.

B.12. Second Forward Pass: Stability During Training

To derive stability constraints for the second forward pass that ensure all intermediate activations are exactly constant scale and the logits are at most constant scale, we will proceed starting from the embedding layer l, followed by the hidden layers l in $[2, L]$ and finally the readout layer $L+1$. These constraints work iteratively across layers: the first constraints will ensure $z_1 = \Theta(1)$, and then the subsequent constraints will ensure $z_l = \Theta(1)$ assuming that $z_{l-1} = \Theta(1)$, and finally the readout constraints will ensure that $z_{L+1} = O(1)$ assuming $z_L = \Theta(1)$.

In the second forward pass, the embedding layer activations are

$$\begin{aligned} z_1^1 &= n^{-a_1}(W_1^0 + \Delta W_1^1)x \\ &= n^{-a_1} W_1^0 x + n^{-a_1}\Delta W_1^1 x \\ &= z_1^0 + n^{-a_1}\Delta W_1^1 x. \end{aligned}$$

Recall that the input x is $O(1)$ and that the input dimensionality d that is the interior dimension in the $W_1 \cdot x$ term is $O(1)$. Since $z_1^0 = \Theta(1)$ by the stability at initialization constraints, we have $\Delta z_1^1 = n^{-a_1}\Delta W_1^1 x = O(1) \iff z_1^1 = \Theta(1)$. Then by plugging in ΔW_1^1 for each optimizer from §B.9, we have

	SGD	Adam	Adafactor
$\Delta W_1^1 \sim$	$n^{-a_{L+1}-b_{L+1}-a_l-c_l}$	n^{-c_1}	$n^{-b_1-c_1}$
$\Delta z_1^1 = n^{-a_1}\Delta W_1^1 x \sim$	$n^{-a_{L+1}-b_{L+1}-2a_1-c_1}$	$n^{-a_1-c_1}$	$n^{-a_1-b_1-c_1}$
$\Delta z_1^1 = O(1) \Leftrightarrow$	$a_{L+1}+b_{L+1}+2a_1+c_1 \geq 0$	$a_1 + c_1 \geq 0$	$c_1 \geq 0$ since $a_1 + b_1 = 0$

Next, for the hidden layer activations we have

$$\begin{aligned} z_l^1 &= n^{-a_l} W_l^1 z_{l-1}^1 = n^{-a_l}(W_l^0 + \Delta W_l^1)(z_{l-1}^0 + \Delta z_{l-1}^1) \\ &= n^{-a_l} W_l^0 z_{l-1}^0 + n^{-a_l} W_l^0 \Delta z_{l-1}^1 + n^{-a_l}\Delta W_l^1 z_{l-1}^0 + n^{-a_l}\Delta W_l^1 \Delta z_{l-1}^1 \end{aligned}$$

where $z_l^0 = \Theta(1)$ by the stability at initialization constraints and we assume that $z_{l-1}^1 = \Theta(1)$ by these constraints on the previous layer. This gives us four terms to bound, and in the table below we write one row for each term and in the columns we write the constraints needed to bound that term for the relevant optimizer.

	SGD	Adam	Adafactor
$n^{-a_l} W_l^0 z_{l-1}^0$	$a_l + b_l - 1/2 = 0$ by stability at init so no constraint required		
$n^{-a_l} W_l^0 \Delta z_{l-1}^1$	$1/2 + r_{l-1} - \omega_l \geq 0$		
$n^{-a_l} \Delta W_l^1 z_{l-1}^0$	$a_{L+1} + b_{L+1} + 2a_l + c_l - \alpha_l \geq 0$	$a_l + c_l - \alpha_l \geq 0$	$1/2 + c_l - \alpha_l \geq 0$
$n^{-a_l} \Delta W_l^1 \Delta z_{l-1}^1$	$a_{L+1} + b_{L+1} + 2a_l + c_l + r_{l-1} - u_l \geq 0$	$a_l + c_l + r_{l-1} - u_l \geq 0$	$1/2 + c_l + r_{l-1} - u_l \geq 0$

Finally, for the logits we have

$$\begin{aligned} z_{L+1}^1 &= n^{-a_{L+1}} W_{L+1}^1 z_L^1 = n^{-a_{L+1}} (W_{L+1}^0 + \Delta W_{L+1}^1)(z_L^0 + \Delta z_L^1) \\ &= n^{-a_{L+1}} W_{L+1}^0 z_L^0 + n^{-a_{L+1}} W_{L+1}^0 \Delta z_L^1 + n^{-a_{L+1}} \Delta W_{L+1}^1 z_L^0 + n^{-a_{L+1}} \Delta W_{L+1}^1 \Delta z_L^1 \end{aligned}$$

where $z_L^0 = \Theta(1)$ by stability at initialization and $z_L^1 = \Theta(1)$ by the constraints on the hidden layers, and we want to find the constraints so that $z_{L+1}^1 = O(1)$.

Similar to the hidden activations, we have four terms to bound and show the constraints for each term and optimizer in the following table:

	SGD	Adam	Adafactor
$n^{-a_{L+1}} W_{L+1}^0 z_L^0$	$a_{L+1} + b_{L+1} - 1/2 \geq 0$ by stability at init so no constraint required		
$n^{-a_{L+1}} W_{L+1}^0 \Delta z_L^1$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \geq 0$		
$n^{-a_{L+1}} \Delta W_{L+1}^1 z_L^0$	$2a_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$	$a_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$	$a_{L+1} + b_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$
$n^{-a_{L+1}} \Delta W_{L+1}^1 \Delta z_L^1$	$2a_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$	$a_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$	$a_{L+1} + b_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$

B.13. Third and Subsequent Forward Passes: Stability During Training

For the third and subsequent forward passes, there are slight modifications required to the stability constraints from the second forward pass. Since we require the activations to be exactly constant scale at initialization, the parameter updates for the embedding and hidden layers are never larger in scale than the initial parameters and therefore never dominate the contribution from the initial parameters to the activations following that layer. However, the readout parameters might have updates that are larger in scale than the initialization, so we need to calculate the scale of the readout parameters after the first update and then consider how this changes the constraints on each optimizer.

For SGD, after the first update we have $W_{L+1}^1 = W_{L+1}^0 + \Delta W_{L+1}^1 \sim \max(-b_{L+1}, -a_{L+1} - c_{L+1})$. This changes the gradients for all layers before the readout layer, which were $g_l^0 = a_{L+1} + b_{L+1} + a_l$, and are now $g_l^1 = \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1}) + a_l$. We account for this by replacing the constraints

$$\begin{cases} g_1^0 + a_1 + c_1 = a_{L+1} + b_{L+1} + 2a_1 + c_1 \geq 0 \\ g_l^0 + a_l + c_l - \alpha_l = a_{L+1} + b_{L+1} + 2a_l + c_l - \alpha_l \geq 0 \\ g_l^0 + a_l + c_l + r_{l-1} - u_l = a_{L+1} + b_{L+1} + 2a_l + c_l + r_{l-1} - u_l \geq 0 \end{cases}$$

with

$$\begin{cases} g_1^1 + a_1 + c_1 = \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1}) + 2a_1 + c_1 \geq 0 \\ g_l^1 + a_l + c_l - \alpha_l = \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1}) + 2a_l + c_l - \alpha_l \geq 0 \\ g_l^1 + a_l + c_l + r_{l-1} - u_l = \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1}) + 2a_l + c_l + r_{l-1} - u_l \geq 0 \end{cases}$$

For Adam, even if the readout parameters do increase in scale after initialization, leading to increased gradient scales, the Adam update scale does not depend on the gradient scale so the existing constraints are sufficient.

For Adafactor, similar to Adam we do not require an additional constraint as a result of a change in gradient scales, but there is one additional constraint required due to the parameter scaling: we require $c_{L+1} \geq 0$ to avoid exponential growth as $n^{-c_{L+1} \cdot t}$ across steps t.

Finally, by induction over the steps, combining all the above constraints ensures stability for any time $t \leq T$. Note that it is essential that we assumed the number of training steps T is $O(1)$ so that this induction step does not introduce any width dependence.

B.14. Nontriviality

Recall that a parameterization is nontrivial if the change in logits after initialization is at least constant scale. This corresponds to exact equality on one of the stability constraints on the logits, specifically

SGD	Adam	Adafactor
$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$
or	or	or
$2a_{L+1} + c_{L+1} - \alpha_{L+1} = 0$	$a_{L+1} + c_{L+1} - \alpha_{L+1} = 0$	$a_{L+1} + b_{L+1} + c_{L+1} - \alpha_{L+1} = 0$
or	or	or
$2a_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$	$a_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$	$a_{L+1} + b_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$

B.15. Summary of Constraints

In Table 2, we summarize the full set of stability and nontriviality constraints derived in the previous sections, which define the alignment-general space of parameterizations.

Table 2. Summary of stability and nontriviality constraints for our alignment-general space of parameterizations.

	SGD	Adam	Adafactor
Stability at init	$a_1 + b_1 = 0$ $a_l + b_l = 1/2$ for $l \in [2, L]$ $a_{L+1} + b_{L+1} \geq 1/2$		
Stable activations	$g_1 + a_1 + c_1 \geq 0$ $g_l + a_l + c_l - \alpha_l \geq 0$ $g_l + a_l + c_l + r_{l-1} - u_l \geq 0$ $1/2 + r_{l-1} + \omega_l \geq 0$, where $g_l \coloneqq \max(a_{L+1} + b_{L+1}, 2a_{L+1} + c_{L+1})$	$a_1 + c_1 \geq 0$ $a_l + c_l - \alpha_l \geq 0$ $a_l + c_l + r_{l-1} - u_l \geq 0$ $1/2 + r_{l-1} + \omega_l \geq 0$	$c_1 \geq 0$ $1/2 + c_l - \alpha_l \geq 0$ $1/2 + c_l + r_{l-1} - u_l \geq 0$ $1/2 + r_{l-1} + \omega_l \geq 0$
Stable logits	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \geq 0$ $2a_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$ $2a_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \geq 0$ $a_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$ $a_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} \geq 0$ $a_{L+1} + b_{L+1} + c_{L+1} - \alpha_{L+1} \geq 0$ $a_{L+1} + b_{L+1} + c_{L+1} + r_L - u_{L+1} \geq 0$ $c_{L+1} \geq 0$
Nontriviality	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$ or $2a_{L+1} + c_{L+1} - \alpha_{L+1} = 0$ or $2a_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$ or $a_{L+1} + c_{L+1} - \alpha_{L+1} = 0$ or $a_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$	$a_{L+1} + b_{L+1} + r_L - \omega_{L+1} = 0$ or $a_{L+1} + b_{L+1} + c_{L+1} - \alpha_{L+1} = 0$ or $a_{L+1} + b_{L+1} + c_{L+1} + r_L - u_{L+1} = 0$

B.16. Tensor Programs as a Special Case

When we assume $\alpha_l = 1 \; \forall l \in [2, L+1]$, $\omega_l = 1/2$ for $l \in [2, L]$, and $\omega_{L+1} = 1$, by plugging these values into our constraints we recover exactly the stability and nontriviality constraints in Yang & Hu (2021); Yang & Littwin (2023). These assumptions are the necessary and sufficient conditions to recover their constraints exactly. In particular, their constraints imply no assumption on u_l as their $\alpha_l = 1$ is maximal so $\alpha_l \geq u_l$ in all cases and the $\Delta z_{l-1} \Delta W_l$ term never dominates the $z_{l-1} \Delta W_l$ term.

B.17. Maximum Stable Learning Rates for All Parameterizations

In Table 1 (repeated here), we compute the maximum stable per-layer learning rate exponents under two specific alignment assumptions: "full alignment" where $\alpha_l = u_l = 1$, and "no alignment" where $\alpha_l = u_l = 1/2$, $l \in [2, L+1]$. In both of these settings, we assume $\omega_l = 1/2$, $l \in [2, L+1]$. This ω_{L+1} term is the alignment exponent on the $\Delta z_L W_{L+1}$ term, which quantifies the alignment between parameter updates in earlier layers that contribute to Δz_L and the initialization in the readout layer W_{L+1}. Our $\omega_{L+1} = 1/2$ relaxes the $\omega_{L+1} = 1$ assumption in Yang & Hu (2021); Yang & Littwin (2023).

The maximal learning rate exponents follow by first plugging in the values for α_l, ω_l, and u_l and then solving for the minimal value of c_l (where minimal c_l corresponds to the maximal learning rate, as c_l is the negative exponent) that satisfies the stability constraints in each layer l. Due to the relaxation with $\omega_{L+1} = 1/2$, for all parameterizations and optimizers in both our alignment settings, this results in values of c_l that make $r_l = 0$ for all $l \in [2, L+1]$, indicating that all layers are being updated maximally and that the parameterization is in a feature learning limit.

Table 1 (repeated). Left: Parameterizations and gradients at initialization for width n. Middle: Max stable per-layer learning rate scaling for each optimizer assuming $\alpha_l = 1, \omega_l = 1/2$ for all layers l. Right: Max stable learning rates assuming $\alpha_l = \omega_l = u_l = 1/2$ for all layers.

		Initialization Variance	Parameter Multiplier	Gradient	SGD LR, Full Align	Adam LR, Full Align	Adafactor LR, Full Align	SGD LR, No Align	Adam LR, No Align	Adafactor LR, No Align
Standard	Embedding	1	1	$1/\sqrt{n}$	$\sqrt{n}$	1	1	$\sqrt{n}$	1	1
	Hidden	$1/n$	1	$1/\sqrt{n}$	$1/\sqrt{n}$	$1/n$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1
	Readout	$1/n$	1	1	$1/n$	$1/n$	$1/\sqrt{n}$	$1/\sqrt{n}$	$1/\sqrt{n}$	1
NTK	Embedding	1	1	$1/\sqrt{n}$	$\sqrt{n}$	1	1	$\sqrt{n}$	1	1
	Hidden	1	$1/\sqrt{n}$	$1/n$	$\sqrt{n}$	$1/\sqrt{n}$	$1/\sqrt{n}$	n	1	1
	Readout	1	$1/\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	$1/\sqrt{n}$	$\sqrt{n}$	1	1
muP	Embedding	$1/n$	$\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1	1	$1/\sqrt{n}$	1
	Hidden	$1/n$	1	$1/n$	1	$1/n$	$1/\sqrt{n}$	$\sqrt{n}$	$1/\sqrt{n}$	1
	Readout	$1/n$	$1/\sqrt{n}$	$1/\sqrt{n}$	1	$1/\sqrt{n}$	1	1	1	1
MFP	Embedding	1	1	$1/n$	n	1	1	n	1	1
	Hidden	1	$1/\sqrt{n}$	$1/n^{1.5}$	n	$1/\sqrt{n}$	$1/\sqrt{n}$	$n^{1.5}$	1	1
	Readout	1	$1/n$	$1/n$	n	1	1	n	$\sqrt{n}$	1

For standard and NTK parameterizations, our full alignment per-layer learning rate prescriptions differ from prior work, and can attain feature learning. For muP and MFP, our full alignment per-layer learning rates coincide exactly for SGD and Adam in Yang & Hu (2021) and Yang & Littwin (2023) respectively as the ω_{L+1} term does not constrain the learning rates in the embedding and hidden layers in those parameterizations.

We note here that throughout the paper, we consider from a theoretical perspective what the maximum stable learning rate exponents should be, but empirically we are interested in the optimal learning rate. It is not necessarily the case that the maximum stable learning rate and optimal learning rate scale with the same exponents, and future work could more carefully investigate the relationship between these two entities.

C. Experimental Details

All experiments use the NanoDO (Liu et al., 2024) decoder-only Transformer architecture employing learned positional embeddings, pre-layer norm (Xiong et al., 2020), and GeLU nonlinearity (Hendrycks & Gimpel, 2016) with no tying of the embedding and readout parameters. We do not use bias terms for weight parameters or Layernorm, following Chowdhery et al. (2023). Layernorm has a learnable scale parameter. All experiments are implemented in Flax (Heek et al., 2023) on top of JAX (Bradbury et al., 2018) and use Optax optimizers (Babuschkin et al., 2020).

All models are trained on the C4 dataset encoded with the T5 SentencePiece (Kudo & Richardson, 2018) tokenizer (Raffel et al., 2020), with an additional beginning-of-sequence (BOS) token, resulting in the vocabulary size of $V = 32,001$ ($32,000$ original vocabulary + 1 BOS).[2] Training inputs are sequence-packed, while evaluation inputs are padded.

We use a fixed batch size 256, context length 512 and depth $L = 8$ for all experiments. Unless stated otherwise, we use typical default optimizer hyperparameters (for SGD, momentum $m = 0.9$, for Adam, $\epsilon = 10^{-9}$, and for Adafactor $\epsilon = 10^{-30}$). We do not use weight decay except for in the weight decay experiments in Figure 9, and we do not use dropout. The learning rate schedule for all experiments uses linear warmup of $1,000$ steps followed by a cosine decay schedule with initial and final learning rates of 0.0.

The different model sizes considered are listed in Table 3. Specifically, we fix the head dimension $h = 128$ and co-scale the model dimension D, number of heads H and MLP dimension F such that $D = H \times h$ and $F = 4 \times D$ in all models. The resulting number of parameters is approximately $L \times 12D^2 + 2VD$, with exact parameter counts reported in Table 3. The compute optimal experiments include models up to $H = 32$ or $H = 48$, and the fixed (50,000) step experiments include models up to $H = 128$.

For each model size, we sweep the learning rate in increments of $2^{0.25}$ or $2^{0.5}$, with the largest stable learning rate determined by a heuristic: if the learning rate exceeds the optimal learning rate and the eval loss exceeds the minimum eval loss by more than 20% or causes NaNs, we consider the learning rate unstable. We ensured that our learning rate sweeps covered this stability threshold so that the gap between the largest plotted learning rate and smallest unstable learning rate is at most $2^{0.5}$ and in many cases is $2^{0.25}$. The learning rate sweep plots show only the stable learning rates so learning rates larger than the rightmost point in each plot can therefore be considered unstable.

For all optimizers except Adafactor, we use ZeRO3 (Rajbhandari et al., 2020) fully-sharded data parallelism (FSDP). Our FSDP implementation did not work with Adafactor out-of-the-box due to tensor shape mismatches as a result of the factored matrices in Adafactor so we omit it for that optimizer.

Table 3. Model sizes used in experiments.

Number of heads H	Model dimension $D = 128H$	MLP width $F = 4D$	Parameter Counts Embedding	Non-embedding	Total
1	128	512	4,108,928	5,749,504	9,858,432
2	256	1,024	8,217,856	14,644,736	22,862,592
4	512	2,048	16,435,712	41,872,384	58,308,096
6	768	3,072	24,653,568	81,682,944	106,336,512
8	1,024	4,096	32,871,424	134,076,416	166,947,840
12	1,536	6,144	49,307,136	276,612,096	325,919,232
16	2,048	8,192	65,742,848	469,479,424	535,222,272
20	2,560	10,240	82,178,560	712,678,400	794,856,960
24	3,072	12,288	98,614,272	1,006,209,024	1,104,823,296
32	4,096	16,384	131,485,696	1,744,265,216	1,875,750,912
48	6,144	24,576	197,228,544	3,824,357,376	4,021,585,920
64	8,192	32,768	262,971,392	6,709,755,904	6,972,727,296
96	12,288	49,152	394,457,088	14,896,472,064	15,290,929,152
128	16,384	65,536	525,942,784	26,304,413,696	26,830,356,480

[2]Effective vocabulary dimension in experiments is $32,101$ due to 100 unused tokens.

C.1. Parameterization Details

This section includes details about the parameterization implementation for our Transformer model. For the purpose of parameterization, the *embedding* layers include the embeddings, positional embeddings and the Layernorm scale parameter, the *hidden* layers include the MLP layers in the Transformer block, the dense query, key and value layers, and the attention output projection layer, and the *readout* layer is just the readout layer.

We use the variant of muP originally proposed by Yang & Hu (2021), which is also presented in Table 9 of Yang et al. (2022).

When the embedding initialization is a constant (i.e. has zero as the exponent), we use 0.01 for the embedding and positional embedding initialization standard deviation. We otherwise omit constant factors from parameterized quantities unless otherwise specified.

The attention operator contains a tensor contraction between the query and key matrices, which induces another question about alignment: if we assume alignment between the query and key, then we should normalize by the head dimension h and if we do not assume alignment then we should normalize by $\sqrt{h}$ inside the softmax. We follow convention and use $\sqrt{h}$ for standard and NTK parameterizations and h for muP and mean-field. However, we note that due to our fixed head dimension that this difference amounts to only a constant factor.

C.2. Tuning constant per-layer learning rate factors

When tuning the per-layer constant multiplicative factors defined in Section 4.2, we use a Bayesian optimization library (Golovin et al., 2017) to perform a three-dimensional hyperparameter search for $(\gamma_1, \gamma_h, \gamma_{L+1})$, using 800 trials with at most 100 trials in parallel with a range set to $[1e-2, 1e2]$ for each constant. If the optimal values for any of the constants is at or near the edge of the range after this first search, we extend the range of the sweep for that constant to 0.01 and 100x the optimal value found in the original sweep and repeat the same tuning procedure.

Since the eval loss has some noise, we consider all trials that perform within 0.1% relative eval loss of the best trial to be equivalently good, and determine the optimal constants using the average for each constant over this set of best trials.

C.3. Adam-atan2 Code Change

To implement *Adam-atan2* using the jax numpy package, imported here as `jnp`, we change a single line of code to replace the default Adam update rule:

```
m / (jnp.sqrt(v + eps_root) + eps)
```

with the Adam-atan2 update rule:

```
4/jnp.pi * jnp.arctan2(m, jnp.sqrt(v))
```

We found that stretching the arctangent function to extend the region where arctangent is linear improved performance slightly, and used

```
4/jnp.pi * lambda * jnp.arctan2(m, lambda * jnp.sqrt(v))
```

with a value of lambda = 8 for all *Adam-atan2* experiments.

D. Alignment Experiments

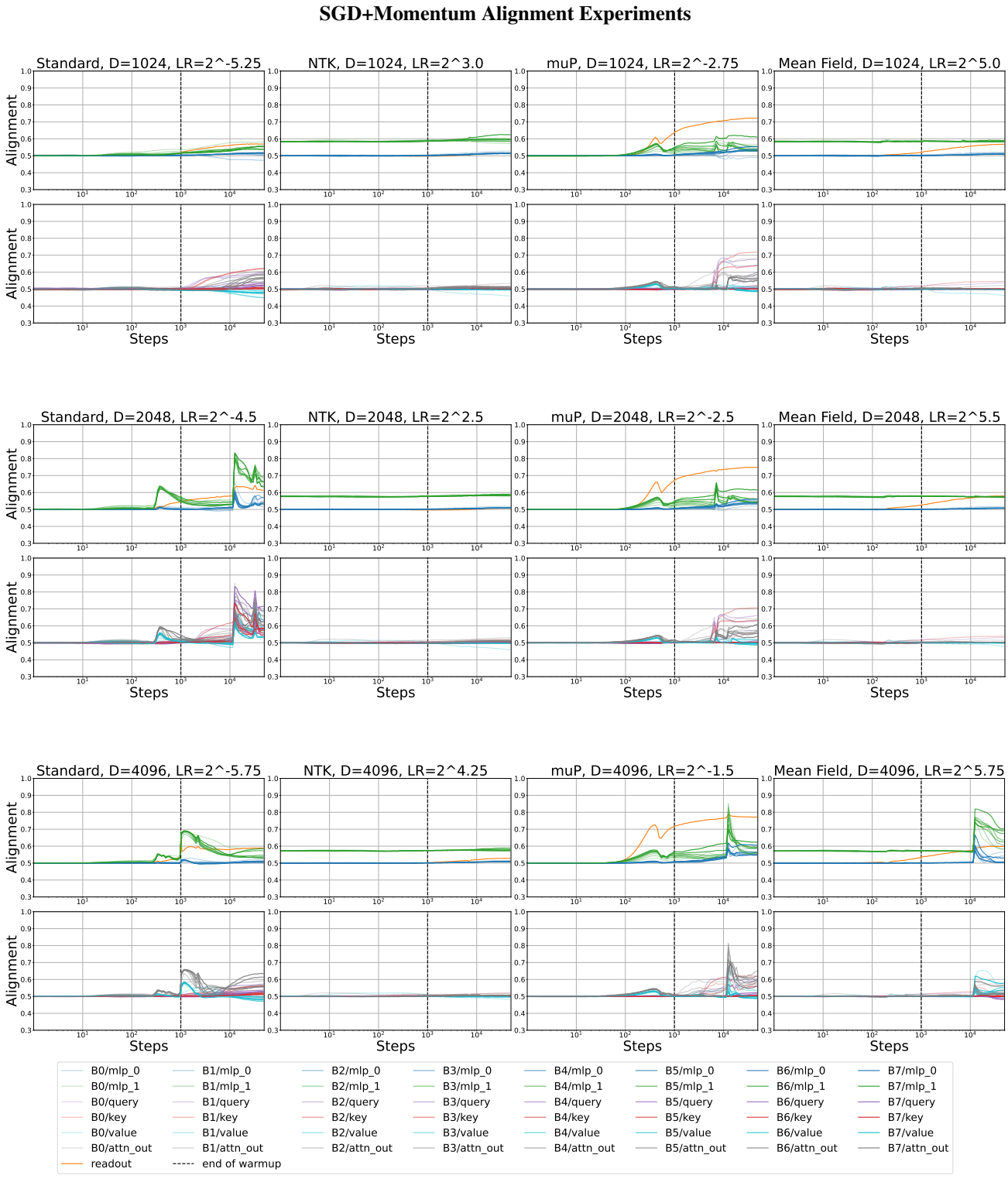

Figure 5. SGD alignment

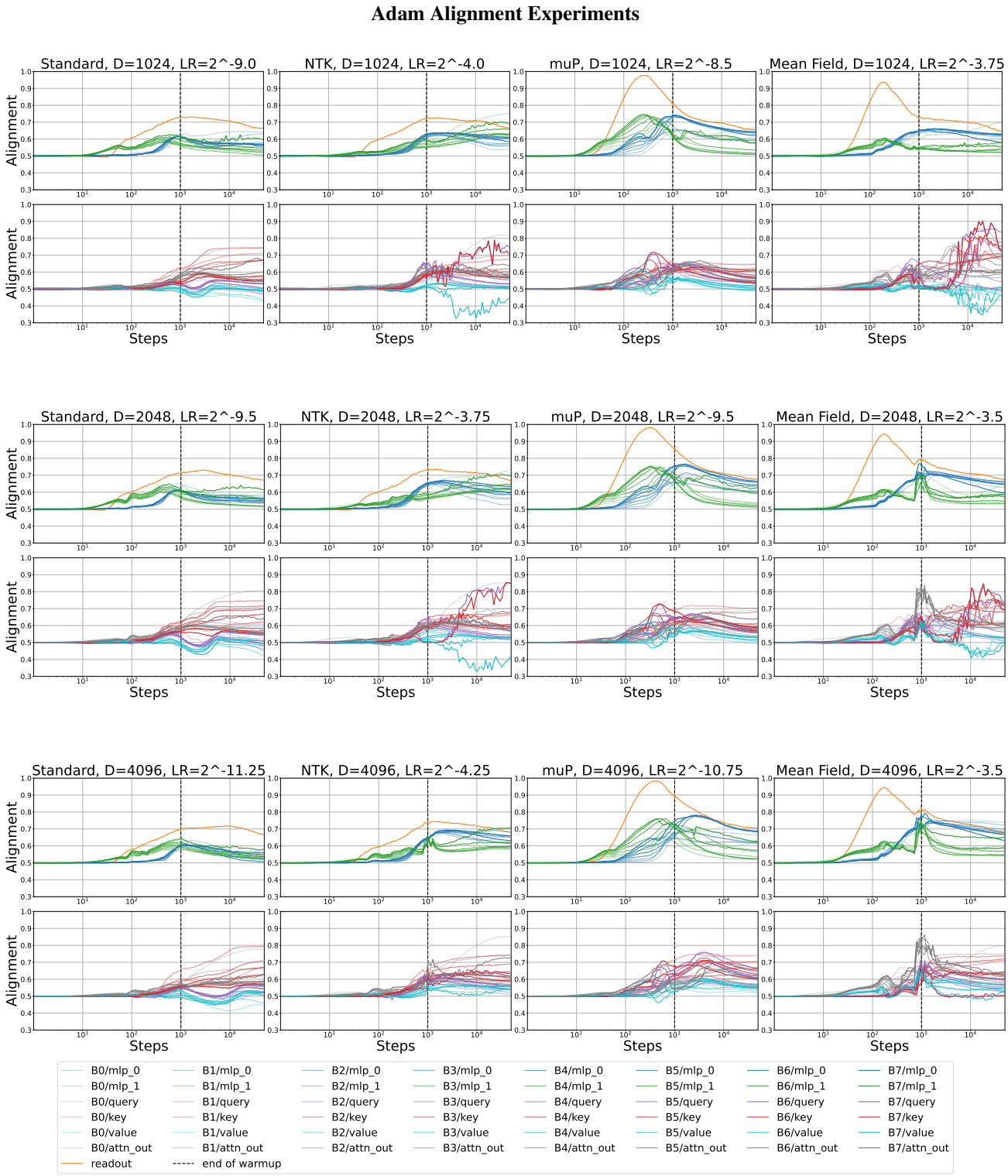

Figure 6. Adam alignment

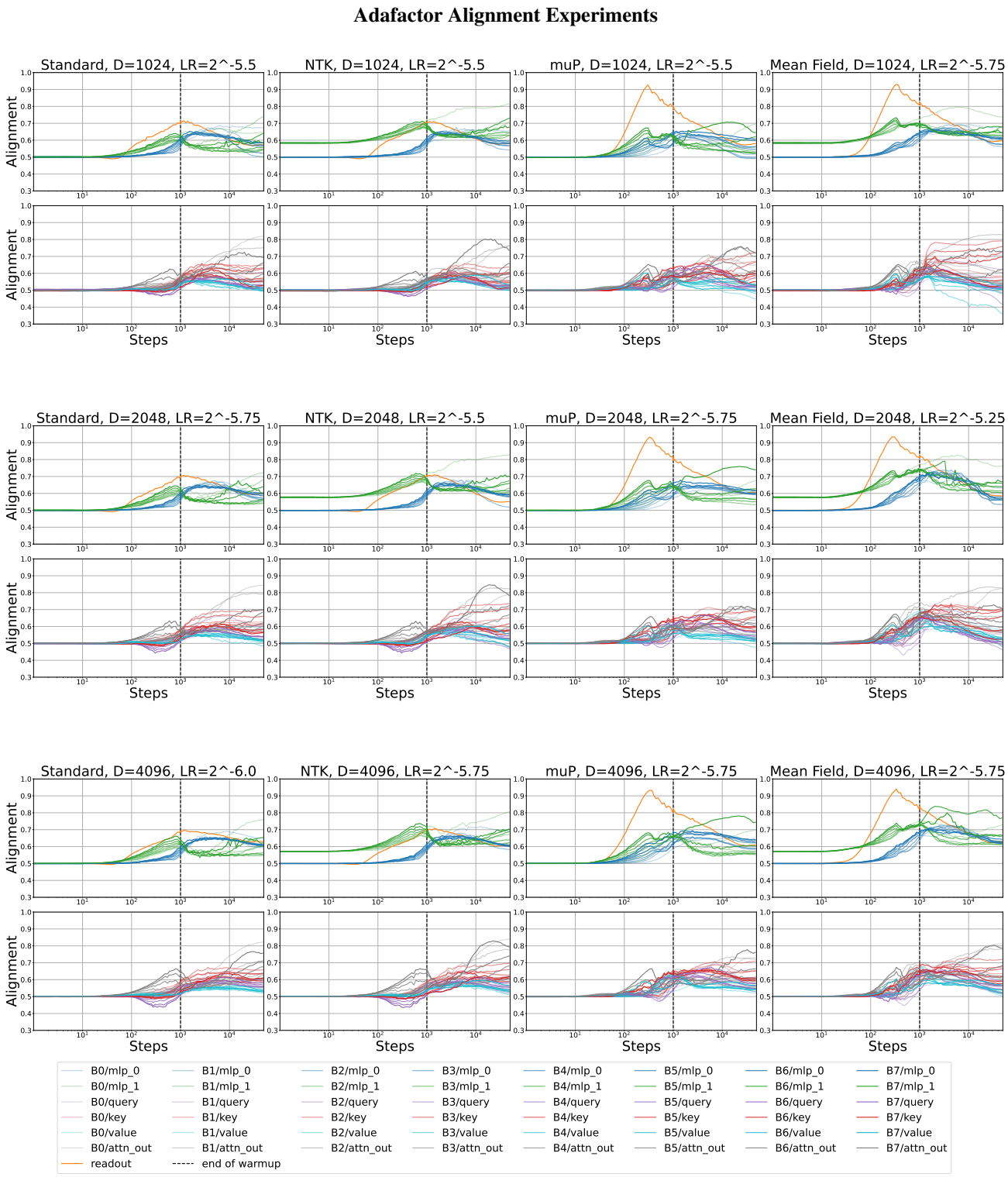

Figure 7. Adafactor alignment

E. Additional Epsilon Experiments

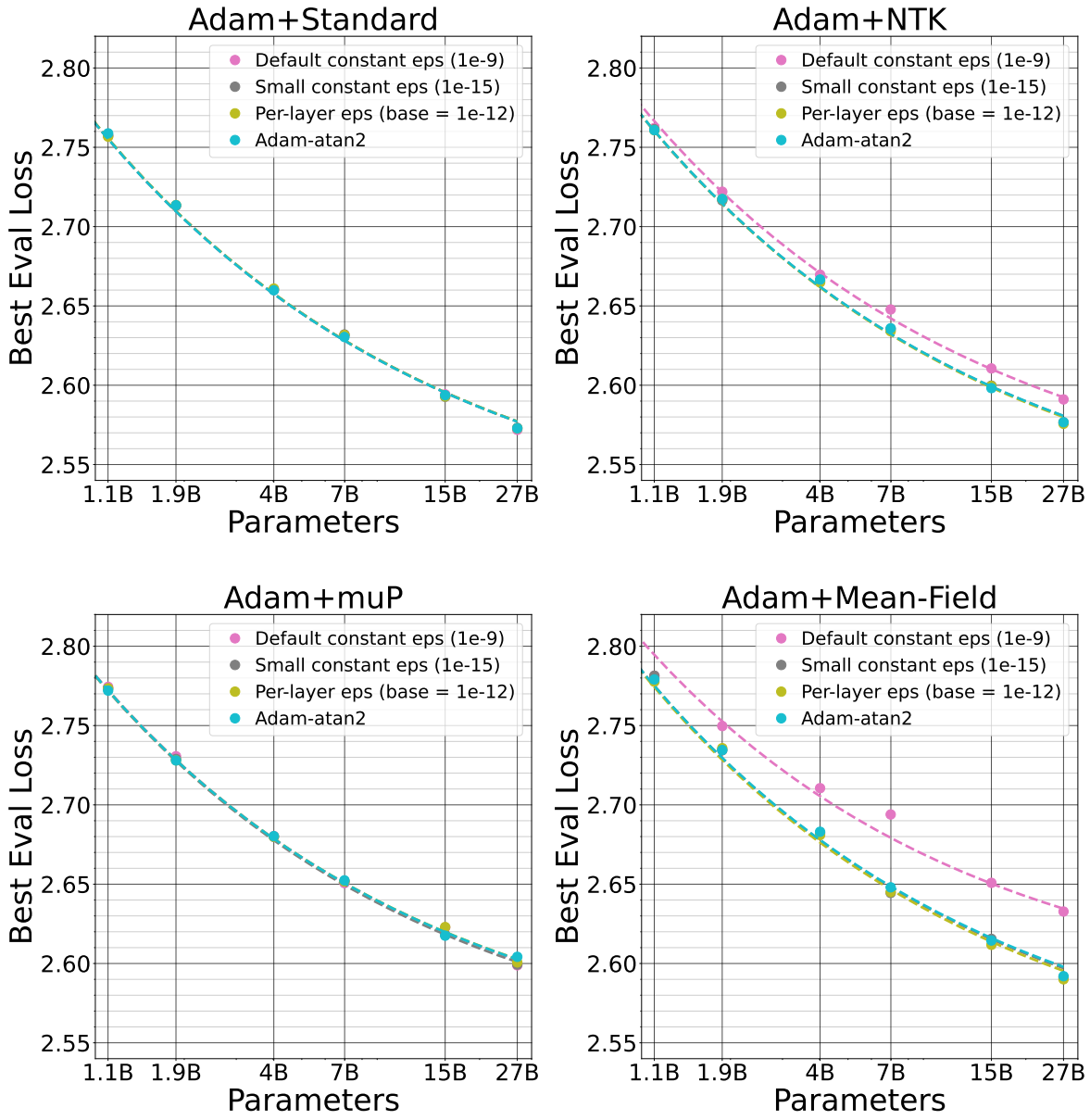

Figure 8. Experiments for all parameterizations comparing the three epsilon mitigations (small constant epsilon = 1e-15, per-layer epsilon with base epsilon = 1e-12, and Adam-atan2) to the baseline default constant epsilon of 1e-9.

F. Weight Decay Experiments

In current practice, weight decay is typically used for training large Transformers and may improve training stability by providing a small amount of regularization (Brown et al., 2020). For the majority of our experiments, we do not use weight decay in order to reduce the number of possible confounding factors and focus our investigation on the impact of the parameterization and optimizer choices.

As a cross-check to ensure our conclusions are likely to transfer to settings with weight decay, we perform a set of experiments for Adam using global learning rates with a small constant weight decay of 1e-4, using "decoupled" or "independent" weight decay as proposed in AdamW (Loshchilov & Hutter, 2018). In decoupled weight decay, the weight decay is not scaled by the base learning rate; our value of 1e-4 decoupled weight decay corresponds to the higher values around 1e-2 or 1e-1 typically used for weight decay that does scale by the base learning rate.

Across parameterizations, we see a slight improvement in the eval loss due to weight decay, but similar learning rate scaling as the zero weight decay setting. This suggests that while weight decay plays a beneficial role, it does not significantly alter the scaling behavior or have parameterization-specific interactions and therefore we expect our conclusions should transfer to settings with a small amount of weight decay. Full results for the weight decay experiments are included in Figure 9.

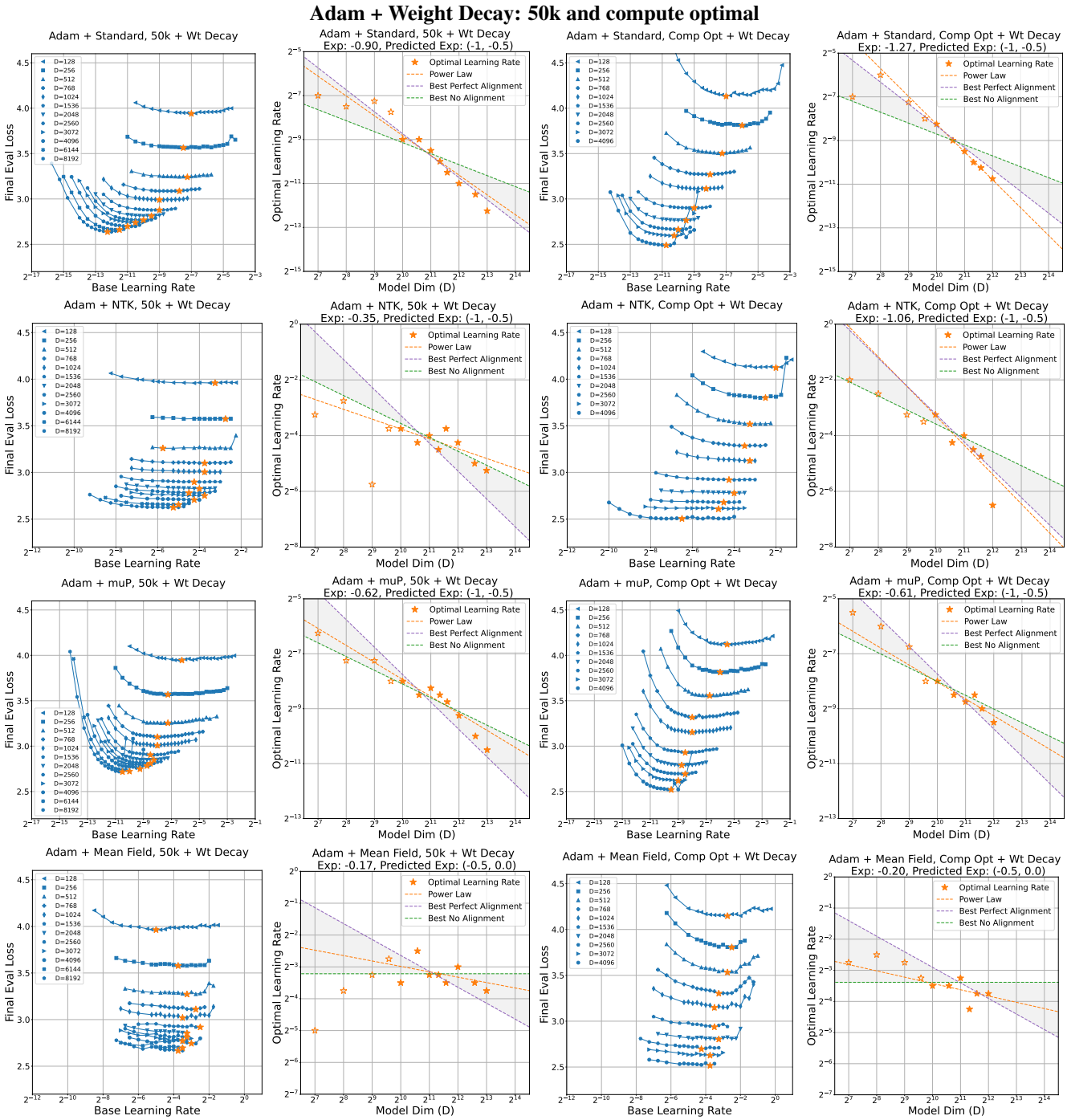

Figure 9. Adam + Weight Decay: 50k and compute optimal

G. Adafactor and Adam + Parameter Scaling experiments

As a cross-check that Adafactor and Adam + parameter scaling are in similar width-scaling regimes, we compare the two optimizers on all parameterizations in two settings: global learning rate + default constants and per-layer learning rate + full alignment + optimal constants. Due to the factored matrices in Adafactor, we encountered issues with tensor shape mismatches when using Adafactor with our implementation of FSDP, which the limited the model sizes we could use for Adafactor. Instead, we use Adam + parameter scaling for all our experiments in Section 4.2. For Adam + parameter scaling, the only difference from our Adam optimizer experiments is the adddition of parameter scaling. We use the standard Optax implementation of Adafactor here that includes the factored second moment estimate and update clipping. We see in Figure 10 that there are minor differences in performance but overall the optimizers show similar scaling behavior across model sizes up to $4B$ parameters, suggesting these two optimizers should be considered members of the same width-scaling regime.

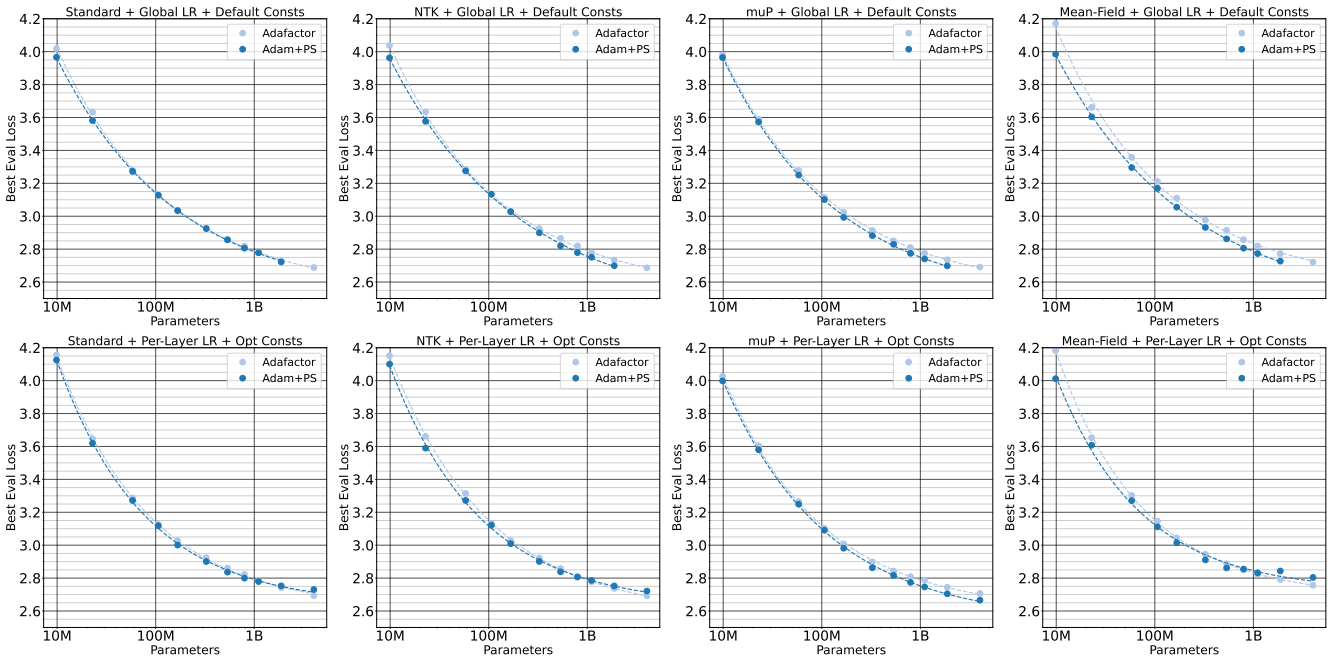

Figure 10. Adafactor and Adam + parameter scaling experiments.

H. Fixed Step vs Compute Optimal experiments

Since the cost of compute is currently the most significant factor that limits the scale of large model training runs, the dominant paradigm for training large models in practice is the compute-optimal regime. The compute-optimal setting (Kaplan, 2019) aims to maximize model performance under a fixed budget of FLOPS for training, where these FLOPS can be traded off between the number of parameters in the model and the number of training tokens the model is trained on. The Chinchilla paper (Hoffmann et al., 2022) finds empirically that the optimal tradeoff occurs when the number of parameters and number of tokens scale in proportion. When the batch size and context length are fixed, as in our setting, the number of training tokens is proportional to the number of training steps. Due to the $n \times n$ parameter matrices in dense hidden layers with width n, the number of parameters grow quadratically with respect to the width. Therefore, the Chinchilla results imply that the compute optimal number of steps grows *quadratically* with respect to the model width.

This contradicts the fixed step assumption used in the theoretical derivations in both this paper and Yang & Hu (2021); Yang & Littwin (2023), which assume that the number of training steps T is $O(1)$. Intuitively, this fixed step assumption is used so that the derivations can consider the contributions to the scaling exponents of a single step at a time: if we satisfactorily bound the contribution of each step to the scaling exponents, and then take only a constant number of steps, then the constant number of steps does not introduce any width-dependent scaling factors. The naive extension of this theory to a setting with $\Theta(n^2)$ instead of $O(1)$ training steps would give impractical bounds: in the worst-case analysis, each learning rate would need to be divided by n^2 to correct for the n^2 number of steps giving learning rates that are far too conservative to be useful.

We therefore take an empirical approach rather than a worst-case theoretical analysis to investigate the role of the training horizon. We perform a set of experiments using both fixed step and compute optimal training horizons in the global learning rate settings for SGD+momentum, Adam and Adafactor across all parameterizations using default constant learning rate multipliers. In each setting, we sweep both model width and learning rate, and then fit a power law with an irreducible loss term to determine the scaling exponent for the optimal learning rate. The measured learning rate exponents are reported in Table 4. For all fixed step experiments, we train for $50,000$ steps. For the compute optimal setting, we compute the training horizon using the Chinchilla-optimal heuristic (Hoffmann et al., 2022) with 20x multiplier, i.e. the number of training tokens is equal to 20 times the number of non-embedding parameters. Full results for the learning rate sweeps are included in Figure 11, 12 and 13.

Table 4. **Optimal global learning rate scaling exponent** with respect to model dimension for given optimizer, parameterization combination. We report measured exponents for fixed step (50k) and compute optimal training horizon settings.

		Fixed Step: 50k	Compute Optimal
SGD	STP	-0.68	-0.91
	NTK	0.65	0.06
	muP	-0.12	-1.17
	MFP	0.25	-0.72
Adam	STP	-1.09	-1.27
	NTK	-0.50	-0.89
	muP	-0.99	-1.24
	MFP	-0.09	-0.49
Adafactor	STP	-0.16	-0.59
	NTK	-0.16	-0.57
	muP	-0.21	-0.62
	MFP	-0.15	-0.51

Our exponent measurements show that in every parameterization $\times$ optimizer setting, the learning rate exponent in the compute optimal setting is *smaller* than in the fixed steps setting, indicating that the learning rate would need to decrease more aggressively as width grows than predictions from the fixed steps setting would imply. The median difference from the twelve optimizer $\times$ parameterization settings is 0.40 and ranges from 0.23 to 1.05. We note this difference of 0.40 is much less than the difference of 2 that would come from the naive worst-case theoretical analysis.

This result has implications both for theoretical and empirical settings. First, it motivates theoretical work to consider the compute optimal setting instead of the fixed steps setting. Second, it implies that hyperparameter search should be careful not to assume that results from the fixed step setting will extrapolate to the compute optimal setting. In particular, given

a fixed compute budget to spend on hyperparameter search before training a single large model with a compute optimal horizon, one possible approach to choosing the learning rate would be to train model sizes close to the final model size for shorter training horizons, and use this to extrapolate the best learning rate for the large model compute optimal run. A priori, this strategy might be advantageous because the shorter training horizons let you use larger models for the same hyperparameter search budget so the search occurs closer in size to the final model. However, our results suggest that this may not be a viable strategy: if we extrapolate a learning rate to larger models based on an exponent fitted in the fixed steps setting, we may significantly overestimate the learning rate that is optimal in the compute optimal setting. Instead, we recommend considering performing the hyperparameter search for the learning rate by training smaller models at their compute optimal training horizon and then extrapolating across model sizes to the compute optimal setting for the largest model.

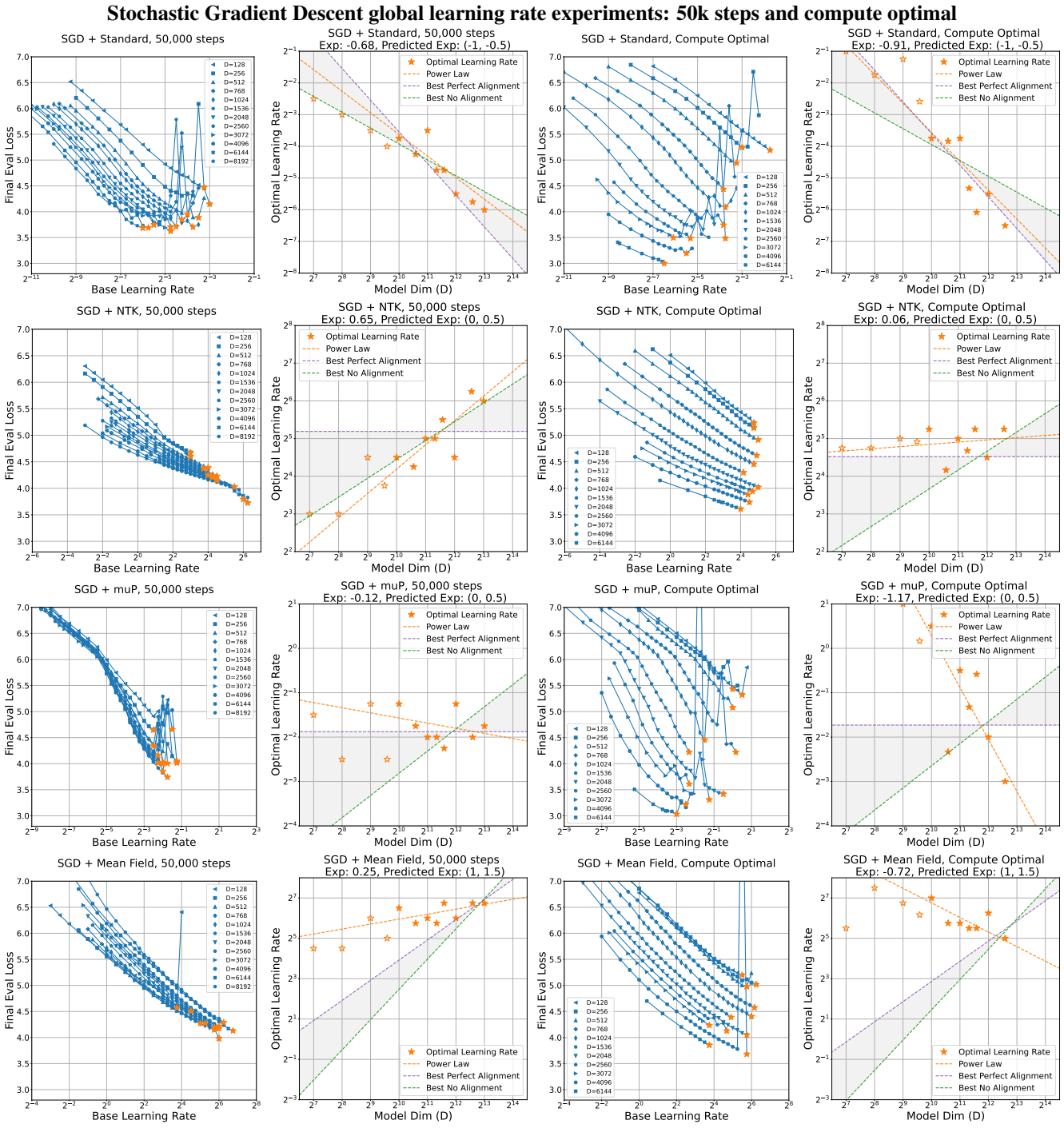

Figure 11. Stochastic Gradient Descent experiments: 50k steps and compute optimal

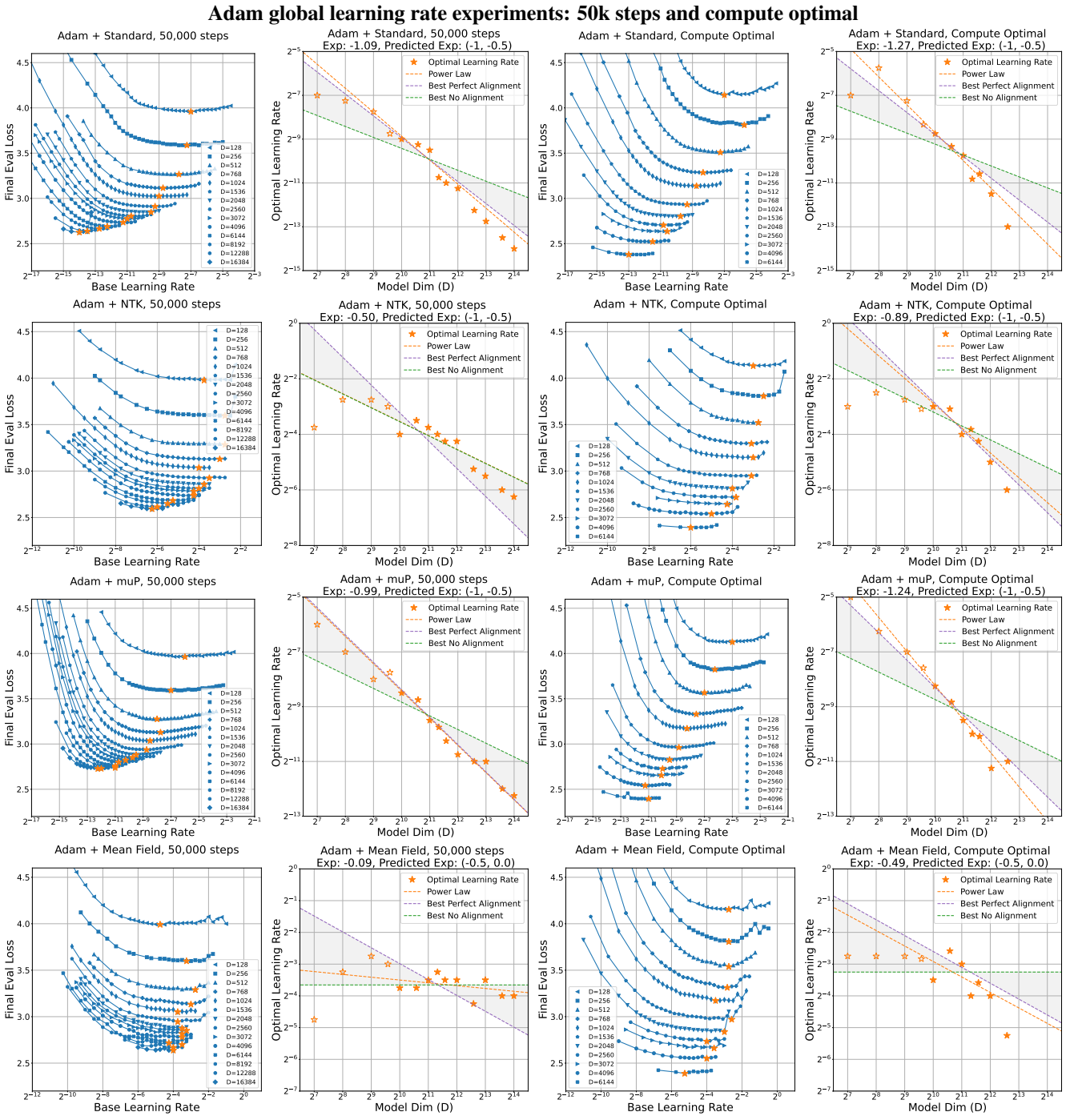

Figure 12. Adam experiments: 50k steps and compute optimal

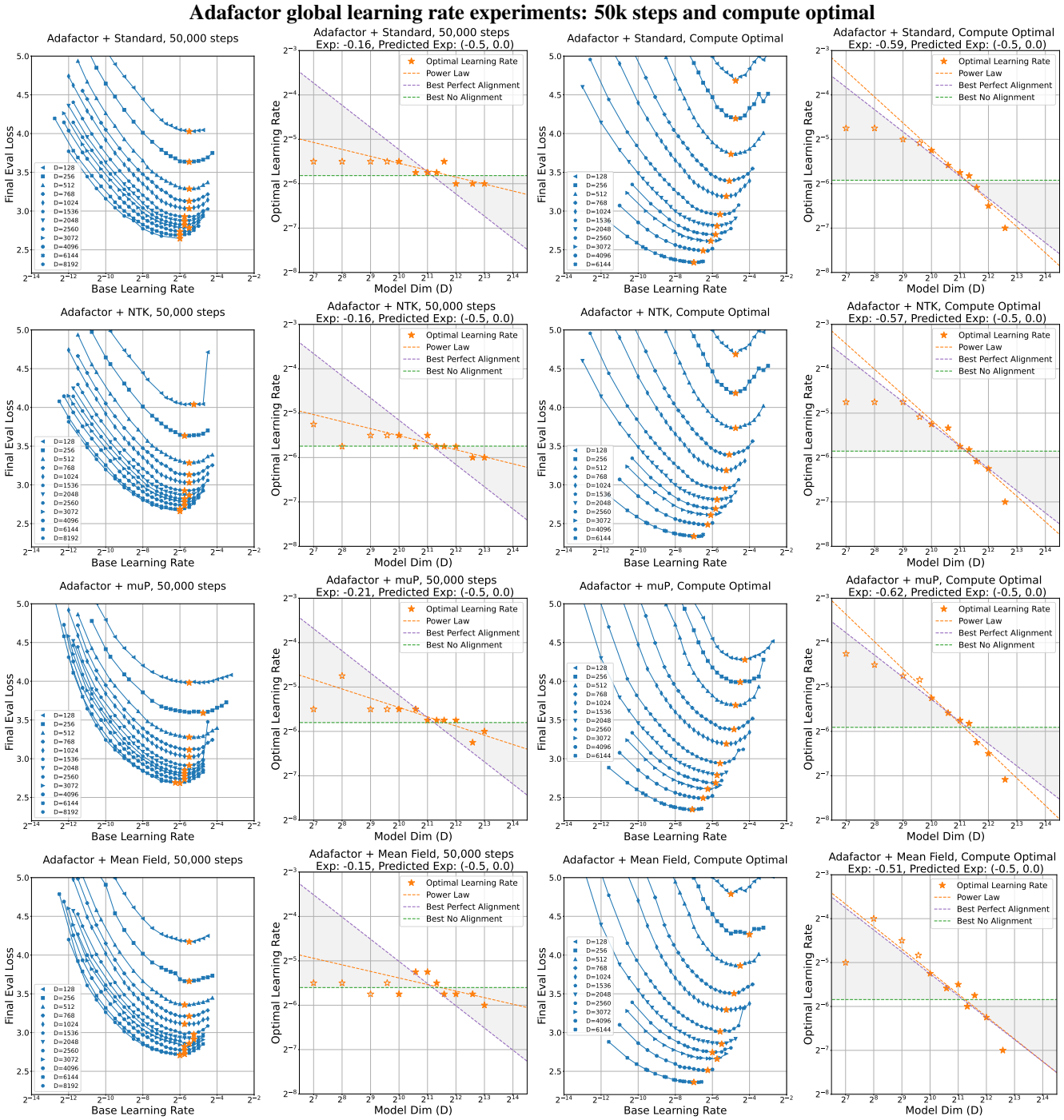

Figure 13. Adafactor experiments: 50k steps and compute optimal