Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, Joel Hestness
Cerebras Systems
{shane.bergsma,joel}@cerebras.net

Abstract

Efficient LLM pre-training requires well-tuned hyperparameters (HPs), including learning rate η and weight decay λ. We study *scaling laws* for HPs: formulas for how to scale HPs as we scale model size N, dataset size D, and batch size B. Recent work [1] suggests the AdamW timescale, $\tau = B/(\eta\lambda D)$, should remain constant across training settings, and we verify the implication that optimal λ scales linearly with B, for a *fixed* N and D. However, as N and D *scale*, we show optimal τ obeys a precise power law in the tokens-per-parameter ratio, D/N. This law thus provides a method to accurately predict λ_{opt} in advance of large-scale training. We also study scaling laws for optimal batch size B_{opt} (the B enabling lowest loss at a given N, D) and critical batch size B_{crit} (the B beyond which further data parallelism becomes ineffective). In contrast to prior work, we find both B_{opt} and B_{crit} scale as power laws in D, independent of model size, N. Finally, we analyze how these findings inform the real-world selection of Pareto-optimal N and D under dual training time and compute objectives.

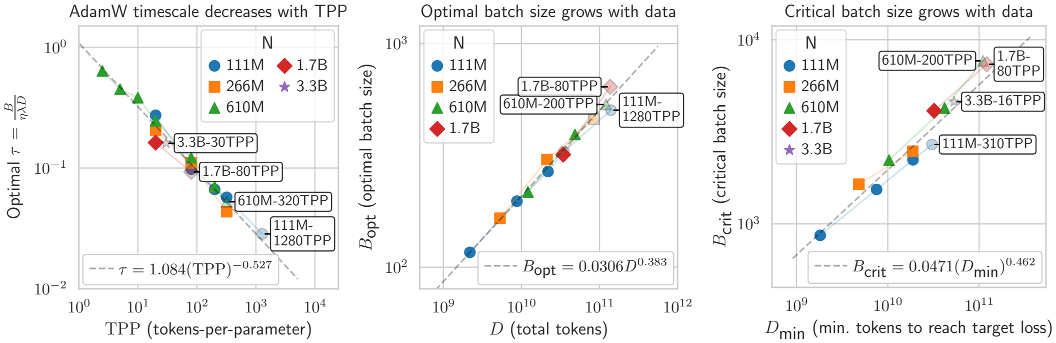


Figure 1: **Hyperparameters and their power lines**: Optimal τ obeys a power law in tokens-per-parameter (*left*), while optimal batch size (*middle*) and critical batch size (*right*) obey power laws in D. Faded markers indicate points not used in fitting; all fits generalize well to larger-scale runs.

1 Introduction

LLMs predictably improve as model size N and training data size D increase [2–4]. Today, state-of-the-art LLMs are trained at computational scales that leave no scope for hyperparameter (HP) tuning, although it is widely accepted that good HPs are critical for effective training [5–7].

Both theoretical and empirical efforts have sought to address this. Theoretically, maximal update parameterization (μP) allows the optimal learning rate η_{opt} and initial weight variance σ^2_{opt} to remain

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

stable when scaling model width [8, 5], enabling a "tune small and train large" strategy. Empirically, DeepSeek LLM [7] adopted "scaling laws for HPs," where optimal batch size B_{opt} and optimal learning rate η_{opt} are estimated at small scale, and then extrapolated via a power law fit in total compute FLOPs, C. A similar approach was used in Kaplan et al. [4], forecasting B_{opt} and η_{opt} from loss L and model size N.

Relying on a unique predicted B_{opt} is *inflexible*—it precludes adjusting B for compute/time trade-offs or hardware constraints. It is also unclear whether C, L, N, or D (or a combination) best explains scaling. [7] notes, "for models with the same [C] but different model/data allocations, the optimal parameter space varies slightly." Also, no comparable study has been done for weight decay λ.

This paper introduces a flexible, unified approach to HPs, using both μP and scaling laws. We fit power laws to losses derived from hundreds of μP-trained models, focusing on combinations of λ, B, N, and D. We study both *compute-optimal* and *overtrained* models. The fewest FLOPs to achieve a loss typically occurs when training at $\approx$20 tokens-per-parameter (TPP = D/N) [3, 9], but overtrained models (>20 TPP) offer more-efficient inference [10]. We study TPPs from 20 to 1280.

To capture λ's interaction with other scaling HPs (η, B), we model scaling of the AdamW timescale, $\tau = B/(\eta\lambda D)$. [1] found optimal τ stable with varying D, but we show it obeys a power law in TPP (Fig. 1, *left*). This law thus enables accurate estimation of λ_{opt} for any N, D, B.

Leveraging these better HPs as B scales, we also study optimal batch size: the B_{opt} that minimizes loss at a given N and D. While B_{opt} scales as a power law in C *when TPP is fixed* (Fig. 5, *left*), our results show this arises from a more fundamental power-law dependence on D (Fig. 1, *middle*).

Importantly, increasing $B > B_{\text{opt}}$ can still reduce training time (fewer steps) and improve hardware utilization. This raises the question: how much *extra* data is needed when using large B? Prior work defines the *critical batch size* B_{crit} as the point where training to a target loss requires $2 \times D_{\text{min}}$, with rapidly-diminishing returns in training speed thereafter [11]. We show B_{crit} also scales with D (Fig. 1, *right*), not L as suggested in [4] (Fig. 5, *middle*), consistent with recent results from [12].

Finally, amid intense competition to advance LLM performance, a key question is: which N, D, and B yield the best trade-off between training speed and compute cost? Using our fit B_{crit} law, we derive Pareto-optimal solutions to these competing objectives, and show that small, overtrained models can be best—offering both faster steps and greater parallelism via larger D (and thus higher B_{crit}).

Key findings and takeaways are highlighted in the paper. Our main contributions are:

- The first large-scale empirical study varying weight decay λ across N, D, and B in LLMs.
- Showing the AdamW timescale obeys a power law, enabling λ_{opt} for any N, D, B (Sec. 2).
- A new method for estimating B_{crit}, suitable for any LR schedule or optimizer (Sec. 3.2).
- Confirmation that both B_{opt} and B_{crit} scale as power laws in D (Sec. 3).
- New methods for selecting N, D, and B to trade-off training time vs. compute (Sec. 4).

2 Scaling of the AdamW timescale τ, and optimal weight decay λ_{opt}

2.1 Background: μP, AdamW, and τ_{epoch}

μP μP is increasingly used in LLM training [13–18]. With μP, base HPs are tuned on a *proxy* model and then transferred to wider [5] and deeper [19] models. Given the width of the proxy model, d_p, and target, d_t, μP prescribes scaling factors to apply to the LR, initial weight variance, and other base HPs. In particular, the optimal base LR, $\tilde{\eta}_{\text{opt}}$ is scaled down to $\eta_{\text{opt}} = ({d_p}/{d_t})\tilde{\eta}_{\text{opt}}$.

While μP enables the same base LR to be used across different N, $\tilde{\eta}_{\text{opt}}$ has empirically been found to *vary with* B [5, 20, 21, 16]. Moreover, recent work has also observed $\tilde{\eta}_{\text{opt}}$ *decreasing in* D, leading to proposals for scaling $\tilde{\eta}_{\text{opt}}$ as a (decreasing) power law in D [16, 22].

The EMA view of AdamW Rather than adjusting η as D scales, Wang and Aitchison [1] proposed that, if using the AdamW optimizer [23] with μP, then the weight decay, λ, should instead be adjusted. To see this, note an AdamW update at each step, t, can be expressed in terms of $\eta\lambda$ as:

$$\theta_t = (1 - \eta\lambda)\theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{1}$$

Here, η is the μP-adjusted LR, and $\hat{m}_t$ and $\hat{v}_t$ are (bias-corrected) exponentially-weighted moving averages (EMAs) of gradients and squared gradients [24]. Wang and Aitchison [1] observed AdamW's parameters, θ_t, can *also* be viewed as an EMA—of weight *updates*. Specifically, the standard EMA form $y_t = (1-\alpha)y_{t-1} + \alpha x_t$ matches AdamW when $y_t = \theta_t$, $\alpha = \eta\lambda$, and $x_t = -\frac{1}{\lambda}\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$. The quantity ${}^1\!/\alpha = {}^1\!/\eta\lambda$ provides a measure of the number of iterations (i.e., steps) over which updates are averaged; [1] denotes it as τ_{iter}. They show that if the timescale is measured in *epochs* as $\tau_{\text{epoch}} = \tau_{\text{iter}}/M$, where M is iterations-per-epoch, then the optimal τ_{epoch} remains stable when N or D scale (on image tasks). I.e., if M scales up, λ should be scaled *down* to maintain constant τ_{epoch}.

2.2 Methods: The AdamW timescale for LLMs, τ, and its scaling

Since LLM pre-training only uses one "epoch" of data, we normalize the timescale as $\tau = \tau_{\text{iter}}/S$, where S is the total number of optimization steps.[1] Moreover, since S=D/B,

$$\tau = \frac{B}{\eta\lambda D} \tag{2}$$

τ reflects the fraction of past iterations to include in the final weights. While Wang and Aitchison [1] did not vary B, their work suggests this fraction should remain constant as B scales. We hypothesize that when moving from compute-efficient to overtrained LLMs, updates can be integrated over a smaller fraction of the data; specifically, that τ_{opt} decreases as a power law in TPP $:= {}^D\!/N$:

$$\tau_{\text{opt}}(\text{TPP}) = c_\tau \cdot \text{TPP}^{m_\tau} \tag{3}$$

where c_τ and m_τ are parameters to be fit. Appendix Algorithm 1 summarizes the fitting procedure. Taking the μP-adjusted η as our LR, λ_{opt} can be computed from Eqs. (2) and (3):

$$\lambda_{\text{opt}} = \frac{B}{\eta \cdot D \cdot \tau_{\text{opt}}(\text{TPP})} = \frac{B \cdot \text{TPP}^{-m_\tau}}{c_\tau \cdot \eta \cdot D} \tag{4}$$

2.3 Experimental details

We use a GPT2-like LLM [25], with ALiBi embeddings [26] and SwiGLU [27]. We train on SlimPajama [28] and always evaluate over a held-out set of 1.1B tokens. We use AdamW and μP, with μP HPs derived from a smaller proxy model, and a linear LR schedule, with a 10% warmup followed by decay-to-zero [29]. Appendix C has full experimental details.

2.4 Results: τ and scaling λ

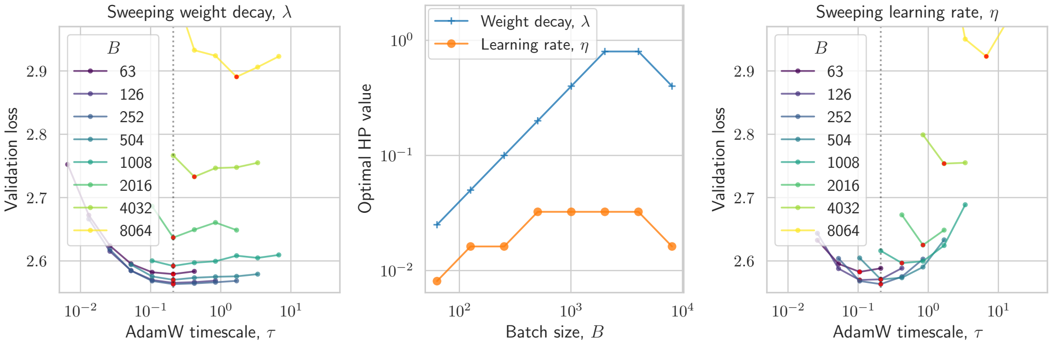


Figure 2: (610M 20TPP): For each B, we sweep λ and find τ_{opt} (*left*). τ_{opt} is stable around 0.21 for $B \in [63, 2016]$, meaning λ_{opt} scales linearly with B over this range (*middle*). When sweeping η (*right*), the lower boundary over all curves is a bowl with a minimum at 0.21; the smallest B settings have τ_{opt} within 2× of this value, but as B increases, τ_{opt} quickly drifts higher.

[1] Because it depends on η, timescale τ *varies* during LR decay. Here we compute τ at peak LR. Notably, if two training runs share the same LR schedule (shape) and τ (at peak LR), their final AdamW timescale (EMA contributions over the *data*) will match—even if B differs. Appendix E.4 provides further details.

Table 1: (610M, 20TPP) Validation losses comparing tuning λ vs. η across B (data from Fig. 2).

$\tilde{\eta}$	λ	$B=$ 63	126	252	504	1008	2016	4032	8064
1.6e-02	0.1	2.595	2.570	**2.563**	2.573	2.599	2.649	2.755	2.923
Tuned	0.1	2.583	2.570	**2.563**	2.571	2.597	**2.625**	2.754	2.923
1.6e-02	Tuned	**2.579**	**2.565**	**2.563**	**2.570**	**2.592**	2.637	**2.733**	**2.891**

***Finding 1**: The optimal τ remains stable as B scales; λ_{opt} scales linearly with B (Fig. 2).*

As B increases and λ is tuned, we find that τ_{opt} remains roughly constant—i.e., changes in B lead to commensurate changes in λ_{opt} (Fig. 2, *middle*)—but only up to a certain point, after which τ_{opt} begins to drift. The drift point corresponds to the critical batch size B_{crit}, above which gradient information no longer scales linearly with B and diminishing returns set in (Sec. 3).[2]

If η is tuned instead, η_{opt} fails to scale with B up to B_{crit} (Fig. 2, *middle*); we observe instead a certain maximum η, above which training becomes unstable; above this, loss spikes occur from which training does not recover. Consequently, η has less flexibility to scale with B; training stability is more fundamental than timescale.

In general, LLMs typically train faster and utilize hardware better with larger B, but only up to B_{crit}: beyond this point, much more data (and compute) is needed to obtain the same loss, without meaningfully reducing the total number of sequential training steps (the poor trade-off for $B > B_{\text{crit}}$ is depicted in Fig. 4). Furthermore, Sec. 3 will show that there is also an *optimal* batch size, B_{opt}, below which loss is worse and utilization/parallelism suffer (because batches are small). In practice, LLMs should therefore be trained in the regime $B_{\text{opt}} \leq B \leq B_{\text{crit}}$. Notably, this is precisely the range where we have shown weight decay to scale predictably with batch size; our findings therefore support the direct optimization of weight decay in the most practically relevant training regimes.

***Finding 2**: With AdamW, we should adjust λ, not η, as B changes.*

Since tuning at scale is infeasible, we need a recipe for selecting HPs in advance. Unlike η, optimal λ follows a predictable relationship with B (Fig. 2, *middle*), making λ the more *viable* target for real-world adjustment. Moreover, since η has less flexibility to maintain optimal timescale, we hypothesize adjusting λ could also be more *effective*. We tested this by comparing either tuning η (using a default λ=0.1—standard practice in LLM pre-training [3, 32–34])—or tuning λ (using the μP proxy-tuned η). Tuning λ was strictly superior in 6 of 8 cases (Table 1).

We also compared adjusting λ versus η as D changes. For a 111M 200TPP model, default HPs obtain a loss of 2.810, tuning η achieves 2.808, and tuning λ obtains 2.805. While differences are small, the key point is that, when scaling B or D: optimizing λ alone is viable and effective.

***Finding 3**: τ_{opt} decreases as a power law in TPP; the law holds at scale (Fig. 1,* left*).*

At each N and D, we calculated τ_{opt} over all (λ, B) pairs; we then fit Eq. (3) to the results. Full details are in Appendix E.2. A precise power law emerges (R^2=0.975), with an optimal τ around 1.0 at 1 TPP, decreasing to 0.01 at 1000 TPP. 10th and 90th percentiles of fitted m_τ over all points are (-0.529, -0.507) (computed as in [3] by bootstrapping: re-fitting on 80% of points, 1000×), indicating a reliable power-law trend. A decreasing τ stands in contrast to the prescription of Wang and Aitchison [1] (for multi-epoch training), who advocated keeping τ constant as D changes.

Fig. 1 (*left*) includes four (labeled) points not used in fitting, but computed later to evaluate predictive ability. Even though some of these points are *interpolated* in terms of the fitting range (i.e., TPP range), they nevertheless represent much greater scales; e.g., training a 3.3B-30TPP model requires 1000× the FLOPs compared to training the 111M-20TPP model (whose data point is plotted nearby). In other words, the law generalizes across at least 3 orders of magnitude in compute.

[2] Training with $B > B_{\text{crit}}$ can be viewed as training with less *effective* data, thus decreasing the TPP, and shifting τ_{opt} higher (by our power law) and thus λ_{opt} *lower*. A decrease in the optimal learning rate η_{opt} when $B > B_{\text{crit}}$ has been previously observed [30, 31]. These works measure η_{opt} at very large batch sizes, e.g., 10× or 100× B_{crit}, which was not practical at the scales that we trained. Nevertheless, our observation of a similar "surge" phenomenon with λ_{opt}, occurring in AdamW rather than vanilla Adam, motivates further study.

Discussion The τ scaling law also predicts previously-observed HP scaling in the literature. E.g., for a fixed N, Eqs. (2) and (3) together imply: $\eta_{\text{opt}} = B \cdot c_{\eta_D} \cdot D^{m_{\eta_D}}$, where c_{η_D} and m_{η_D} are parameters. This matches Equation (1) in [16], and our implied m_{η_D} is close to their fit value (see Appendix E.3.1). [22] also scales η as a power law in D. They note that fitted power law exponents are similar when B is doubled, although the optimal η is "higher". More precisely, we can see from their Figure 13 that the optimal η appears to, in fact, also double—consistent with the derived η_{opt} equation above.

Key takeaway 1: *With AdamW, you can find τ_{opt} for a small N, D by tuning λ. From there, τ_{opt} scales $\propto ({}^{D}/{}_{N})^{-0.5}$. At larger N, D, set λ_{opt} via Eq. (4) and enjoy well-tuned models.*

3 Scaling of optimal batch size B_{opt} and critical batch size B_{crit}

We now develop methodology to estimate B_{opt} and B_{crit} over the dimensions of total training tokens D, total compute FLOPs C, and validation loss L. Results show power-law scaling of both B_{opt} and B_{crit} with D, enabling estimation of B_{crit} at scale via a small number of test runs at modest budgets.

3.1 Background: B_{opt} and B_{crit}

B_{opt} As noted above, recent work has pursued an *optimal* B: the B achieving lowest loss given N, D. Hu et al. [17] fit B_{opt} using a power law in (estimated) loss, and use μP to set η. Joint power laws for optimal η and B have also been fit [7, 35, 36]. E.g., [7] estimated $B_{\text{opt}} = 0.292C^{0.3271}$ (in tokens); we refer to this fit as B_{deepseek} below. Li et al. [37] found η_{opt} to scale in N, D, while B_{opt} primarily scales in D. Qwen2.5 [38] also report studying how η_{opt} and B_{opt} scale with N and D (across dense and mixture-of-expert LLMs), but without further details.

B_{crit} Let D be the number of training tokens required to reach target loss $\hat{L}$ when using a batch size of B. $S = D/B$ is the corresponding number of optimization steps. Doubling a small B doubles per-step gradient information; $\hat{L}$ can be reached in half the optimization steps, using the same total D (so-called "perfect scaling" [39, 40]). But as B increases further, step-wise gradient information becomes more and more redundant: eventually, much larger D is required to reach $\hat{L}$, and S decreases only marginally. McCandlish et al. [11] show that $\langle D, S \rangle$ pairs can be well fit by the equation:

$$S/S_{\text{min}} - 1 = (D/D_{\text{min}} - 1)^{-1} \tag{5}$$

where S_{min} and D_{min} are parameters to be fit. Intuitively, $D_{\text{min}} < D$ is the asymptotically minimum number of tokens that can reach $\hat{L}$ (achieved with B_{opt}) and $S_{\text{min}} < S$ is the asymptotically minimum number of *steps* that can reach $\hat{L}$ (achieved as $B \to \infty$). Eq. (5) defines a hyperbolic curve, like those in Fig. 4, where B controls the position on the curve and can be set depending on the importance of time (higher $B \to$ higher D, lower S) or compute (lower $B \to$ lower D, higher S).

Definition 3.1. The *critical batch size* at $\hat{L}$ is defined from the fit of Eq. (5) as B_{crit}=$D_{\text{min}}/S_{\text{min}}$.

From Eq. (5), we can derive (Appendix F.1), for a given B, the D needed compared to D_{min}:

$$D = D_{\text{min}}(1 + B/B_{\text{crit}}) \tag{6}$$

Eq. (6) implies that when B=B_{crit}, we require $2 \times D_{\text{min}}$ tokens (and $2 \times S_{\text{min}}$ steps) to reach $\hat{L}$. B_{crit} is a *transition* point along the D vs. S curve: for $B > B_{\text{crit}}$, much higher D is needed for only small reductions in S (Fig. 4). Kaplan et al. [4] refer to B_{crit} as the *optimal compromise* between time and compute. They determine B_{crit} at smaller scales and fit a power law for B_{crit} as a function of $\hat{L}$.

Eqs. (5) and (6) also imply B_{opt} is theoretically equal to 1. In practice, loss degrades below a particular B_{opt} [17, 7, 35], a finding that "appears to contradict the conventional wisdom" about B_{crit} [35]. With well-tuned λ, we find small differences in loss across small B, suggesting Eq. (5) may nevertheless provide a good fit to observed data. Appendix B has further discussion.

In recent work, Zhang et al. [12] define B_{crit} as the point where $D = 1.2 \times D_{\text{min}}$. [12] uses a different training setup, with a constant LR, weight averaging, and no weight decay. Notably, they observe little change in B_{crit} as N varies at fixed D, but, for a 302M model, find power-law scaling in D as $B_{\text{crit}} = 22.91D^{0.47}$ (in tokens), consistent with observed scaling across models at fixed TPP. See Appendix D.3 for further differences with Zhang et al. [12].

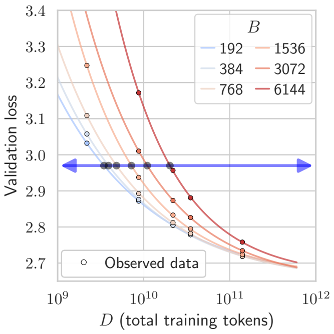


Figure 3: (111M): **Fitted B-specific power laws**, $L_B(D)$, for inferring steps to reach target loss $\hat{L}$ (arrowed blue line).

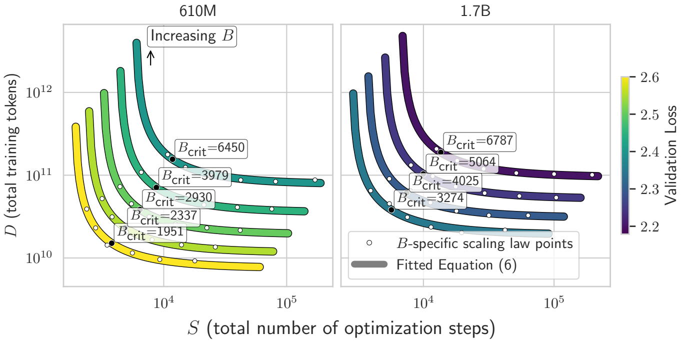


Figure 4: **Eq. (5) fits observed data well**: Increasing B (moving leftward along curves) decreases optimization steps (x-axis), but requires more tokens (y-axis) to reach target loss (in color). B_{crit} is the transition point along each fitted curve.

3.2 Methods: estimating B_{opt} and B_{crit}, and their scaling

Estimating B_{opt} We use the same experimental settings as Sec. 2.3. To ensure good HPs, we sweep λ by factors of $2\times$ at each B, D, N, except at the largest scales (see Appendix Table 3) where we set λ via the projected value from Eq. (4). In all figures, B is reported in units of *sequences*.

Estimating B_{crit} Unlike B_{opt}, measuring B_{crit} requires training models with different B *to the same* $\hat{L}$. Unfortunately, we do not know *a priori* how many steps are required to reach $\hat{L}$, yet we need this information to configure a LR schedule that reaches its minimum value on the final step (the typical setup, shown to be consequential in prior work [3, 41]). Unfortunately, it is not feasible to *search* for the precise steps needed, i.e., by conducting training runs with different schedules/step budgets.

McCandlish et al. [11] address this issue by performing a single training run at a constant LR, while Zhang et al. [12] also use a constant LR, but use weight averaging to frequently generate higher-quality checkpoints for evaluation (Appendix D.3).

In contrast, we desired a method agnostic to the LR schedule. We achieved this by fitting batch-size-specific power laws that model how *loss* scales with D. These laws allow us to accurately *interpolate* the D required to reach $\hat{L}$. Fig. 3 depicts, for different B and a given $\hat{L}$, the interpolated D values (intersection points of arrowed line and fitted loss curves). The full process to obtain B_{crit} at $\hat{L}$ is:

1. For each B, train over different D, and subsequently fit a B-specific power law $L_B(D) = E_N + D_{\text{const}} D^{-\beta}$ on the resulting loss values (fitted curves in Fig. 3).
2. Use fitted $L_B(D)$ to infer the D_B needed to reach $\hat{L}$ as: $D_B = L_B^{-1}(\hat{L}) = \left({}^{D_{\text{const}}}/_{\hat{L}-E_N}\right)^{\frac{1}{\beta}}$.
3. Fit Eq. (5) on the resulting $\langle D_B, S{=}D/B \rangle$ pairs, and obtain $B_{\text{crit}} = D_{\text{min}}/S_{\text{min}}$.

This method makes no assumptions about the LR schedule or optimizer, while enabling measurement of B_{crit} at arbitrary losses without re-training. Appendix F.2 provides further details, including fits of $L_B(D)$ at other model scales (Fig. 9) and a summary of the full procedure (Algorithm 2).

B_{opt} and B_{crit} scaling We collect B_{opt} across different N and D, and fit a power law in both data D and compute C (via the standard approximation $C \approx 6ND$ [4, 3]). For B_{crit}, we use the procedure described above to estimate B_{crit} across multiple $\hat{L}$, across different N. From each B_{crit} estimate, we obtain a pair $\langle D_{\text{min}}, B_{\text{crit}} \rangle$. We propose that B_{crit} follows a power law in D_{min}, according to:

$$B_{\text{crit}}(D_{\text{min}}) = c_{B_{\text{crit}}} \cdot D_{\text{min}}^{m_{B_{\text{crit}}}} \tag{7}$$

Where $c_{B_{\text{crit}}}$ and $m_{B_{\text{crit}}}$ are fit on the $\langle D_{\text{min}}, B_{\text{crit}} \rangle$ pairs.

3.3 Results: B_{opt} and B_{crit}

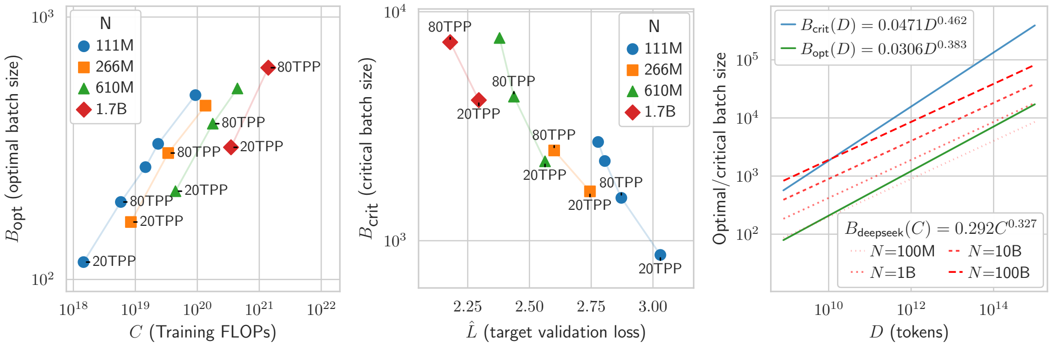


Figure 5: Prior work suggests B_{opt} scales in C (*left*) and B_{crit} in loss (*middle*), but *this only holds at a fixed N/TPP* (same data as Fig. 1 *middle/right*); Fig. 1 shows scaling in D is the fundamental relationship. Plotting $B_{\text{deepseek}}(C)$ law from [7] (*right*), but over D (using $C \approx 6ND$ to obtain C for the spurious dependence on N), we see [7] used generally efficient B values (i.e., within the $B_{\text{opt}} < B < B_{\text{crit}}$ regime) despite fitting C rather than D (B_{opt} and B_{crit} lines from Fig. 1).

***Finding 4**: Eq. (5) provides a decent fit to the trade-off between training time and compute.*

Across different model scales and loss targets, we consistently find that our $\langle D, S\rangle$ pairs fit Eq. (5) well (examples in Fig. 4, appendix Fig. 10). Fits are worse at very small B, as noted above: smaller batches are not monotonically more efficient; Appendix B discusses some potential reasons for this.

***Finding 5**: B_{opt} and B_{crit} obey power laws in D and D_{min}, not in C or L.*

B_{opt} and B_{crit} datapoints fit power laws quite well (Fig. 1, *middle*, R^2=0.984) and (Fig. 1, *right*, R^2=0.940). 10th and 90th percentiles over all points are (0.367, 0.391) for fitted $m_{B_{\text{opt}}}$ and (0.491, 0.526) for $m_{B_{\text{crit}}}$ (computed as in Sec. 2.4). Note $m_{B_{\text{crit}}}$ is higher when fitted over *all* points (as opposed to only small-scale runs), partly reflecting the 111M points trending lower as TPP increases.

Our fitted B_{crit} power law exponent is very close to that from [12]: 0.47 vs. 0.462. Given the many differences in approach (including dataset, use of weight decay, LR schedule, etc., Appendix D.3), this agreement suggests the fundamental relationship of B_{crit} with D persists across such differences.

Fig. 5 (*left*) plots B_{opt} versus C and Fig. 5 (*middle*) gives B_{crit} versus L using the same data as in Fig. 1. In each case, a power law does not fit all points (as proposed previously), but points at the same N, or same TPP, can roughly be linked by (parallel) lines. This is a consequence of power-law scaling in D (see Appendix F.4). That is, scaling in D is the fundamental scaling relationship: B_{opt} and B_{crit} both scale in D regardless of TPP, model size, or loss—it is only when using another (misleading) scaling factor such as C or $\hat{L}$ that TPP or model size appears important, as in these plots.

Fig. 5 (*right*) compares the recommended batch sizes from B_{deepseek} to those from B_{opt} and B_{crit}. Since B_{deepseek} scales in C, it is larger for larger N. Over a range of modern model sizes, B_{deepseek} values generally fall between our projected B_{opt} and B_{crit}, varying in the extent to which they are compute-efficient (close to B_{opt}) or time-efficient (close to B_{crit}).

***Finding 6**: Weight decay affects the accuracy of fitted batch size scaling laws.*

Prior work has typically held λ fixed when fitting batch-size scaling laws [7, 35, 12]. Doing so not only degrades loss (Sec. 2) but also reduces the accuracy and generality of the fitted scaling relationships. We demonstrate this in appendix Table 5: rather than tuning λ for each B, we train with several fixed λ values across all runs. As Table 5 shows, increasing λ systematically raises the estimated B_{opt}. This arises because the fundamental scaling variable is the AdamW timescale $\tau = B/(\eta\lambda D)$: when λ increases, the batch size that minimizes loss must increase proportionally to preserve the optimal τ. In Appendix F.5, we show that these effects distort the fitted power-law slope and reduce fit quality (R^2), leading to scaling laws that do not generalize to large-scale training—even if the same fixed weight decay is used there. When λ is tuned to maintain the optimal timescale (final row of Table 5), the resulting B_{opt} follows a clean and accurate power law.

Similar distortions occur for B_{crit} when λ is fixed (Appendix F.5).

Key takeaway 2*: You can estimate B_{opt} and B_{crit} for a small N by training with different B, D and λ_{opt}, and computing loss. From there, $B_{opt} \propto D^{0.4}$ and $B_{crit} \propto D^{0.5}$. At larger N and D, Eq. (6) lets you estimate trade-offs in FLOPs ($\propto D$) vs. training time ($\propto S = D/B$) at different B.*

4 Training settings for balancing time and compute

4.1 Background: compute-optimal and overtrained models

Given a fixed training FLOPs budget, C, how should we allocate model size N versus number of training tokens D in order to minimize *loss*? Hoffmann et al. [3] propose to model loss as:

$$L(N, D) = E + N_{\text{const}} N^{-\alpha} + D_{\text{const}} D^{-\beta} \tag{8}$$

N_{const}, α, D_{const}, and β are parameters fit on observed training runs. From Eq. (8), [3] derives functions for loss-optimal $N_{\text{opt}}(C)$ and $D_{\text{opt}}(C)$ (constraining $L(N, D)$ by $C \approx 6ND$). Results indicate N_{opt} and D_{opt} scale roughly equally as C increases, with the optimal $^{D}/_{N}$ ratio relatively constant at around 20 TPP. Replication studies have found similar results [9, 35], and 20 TPP has become a rule-of-thumb for compute-optimal training [13, 12].

Overtrained, inference-efficient models [10, 42, 43] have largely trained with similar batch sizes to those used in compute-optimal training; such efforts should now consider training with much greater data parallelism, leveraging our finding that B_{opt} and B_{crit} will be higher given the higher training D.

4.2 Methods: exploring the trade-offs of FLOPs vs. time

To compare models of different sizes on a common temporal axis, we must map number-of-optimization-steps to a common temporal scale. Our initial approximation is Training Time $\propto$ $^{\text{Total FLOPs}}/_{B}$, which is also FLOPs *per token* times number of steps. E.g., if FLOPs $\approx 6ND$, Training Time $\approx 6ND/B = 6N \cdot S$. This aligns well with our measured runtimes: doubling N doubles step time; doubling B halves wall-clock time (for the same S).

Now, assume a model of size N can train to loss $\hat{L}$ using D_{min} tokens (here *min* denotes using B_{opt}). Let us refer to N and D_{min} as a *base setting*. A variety of N, D_{min} pairs can reach $\hat{L}$ in the B_{opt} setting, from small models trained on many tokens, to large models trained on fewer tokens. [3] refers to these as iso-loss *contours* of Eq. (8). Suppose a given base setting requires $C(N, D_{\text{min}})$ FLOPs. From this setting, we may increase B to decrease training time (fewer steps), but Eq. (6) indicates a need for $(1 + B/B_{\text{crit}})$ extra *data* in order to reach the same $\hat{L}$. If FLOPs is linear in D (as in $C = 6ND$), we will require the same proportion of extra FLOPs, i.e.,

$$C_{+}(N, D_{\text{min}}, B) = C(N, D_{\text{min}})(1 + B/B_{\text{crit}}(D_{\text{min}})) \tag{9}$$

where $C_{+}(N, D_{\text{min}}, B)$ denotes the total FLOPs needed at $B > B_{\text{opt}}$, and $B_{\text{crit}}(D_{\text{min}})$ captures that the excess FLOPs depends on B_{crit}, which itself scales with D_{min}. In other words, the base setting dictates B_{crit}, and B/B_{crit} dictates the excess FLOPs.

Consider a target FLOP budget of $C_{+}(N, D_{\text{min}}, B) = \hat{C}$ and the goal of reaching $\hat{L}$ *as fast as possible*. Since time $\propto$ $^{\text{Total FLOPs}}/_{B}$, time is minimized by maximizing B. However, by construction, B is not a free variable: it is constrained by Eq. (9) and can be expressed as a function of N and D_{min}:

$$B(N, D_{\text{min}}) = \left(\frac{\hat{C}}{C(N, D_{\text{min}})} - 1 \right) B_{\text{crit}}(D_{\text{min}}) \tag{10}$$

Time is therefore minimized by finding N, D_{min} that maximize this function (over all the N, D_{min} that train to loss $\hat{L}$). The $\hat{C}/C$ ratio is a measure of the *excess* FLOPs that can be spent toward increasing B; it is largest when $C(N, D_{\text{min}})$ is smallest, i.e., when N and D_{min} is most compute-efficient (i.e., $N/D_{\text{min}} \approx$ 20 TPP). But Eq. (10) as a whole captures an elegant tension between compute efficiency and B_{crit}: we can maximize B (and minimize training time) by either (1) minimizing the FLOPs of the base setting (generating more excess FLOPs for increasing B), or (2) maximizing D_{min} (overtraining, which increases $B_{\text{crit}}(D_{\text{min}})$). For a given $\hat{C}$, either (1) or (2) may take precedence.

We use the following procedure to explore the time vs. compute Pareto frontier for a target loss $\hat{L}$:

1. Fit Eq. (8) on our B_{opt} training runs. Express resulting $L(N, D_{\text{min}})$ as $D_{\text{min}\,\hat{L}}(N)$.
2. Using $D_{\text{min}\,\hat{L}}(N)$, get contour points $\langle N, D_{\text{min}} \rangle$ of the given $\hat{L}$. Each such pair consumes $C(N, D_{\text{min}}) \approx 6ND_{\text{min}}$ FLOPs and takes $^{C(N,D_{\text{min}})}/_{B}$ time (rightmost points on Fig. 6 curves).
3. Use Eq. (9) to compute $C_{+}(N, D_{\text{min}}, B)$ as we scale B (crucially, using the estimate of B_{crit} from fitted Eq. (7)), and generate further points along each curve.
4. The non-dominated points over all curves provide the time vs. compute Pareto frontier.

4.3 Results: balancing time and compute

We carry out this procedure using model sizes of 150M, 210M, 550M, 1.1B, and 2.1B, and a loss target of $\hat{L}$=2.6, yielding iso-loss contour points from 150M 600TPP to 2.1B 2TPP. Our fit of Eq. (8) yielded α=0.313 ≈ β=0.282, giving an optimal TPP ratio of ≈20.6 at $\hat{L}$=2.6.

Finding 7*: Overtrained, but not undertrained, models are on the FLOPs vs. time Pareto frontier.*

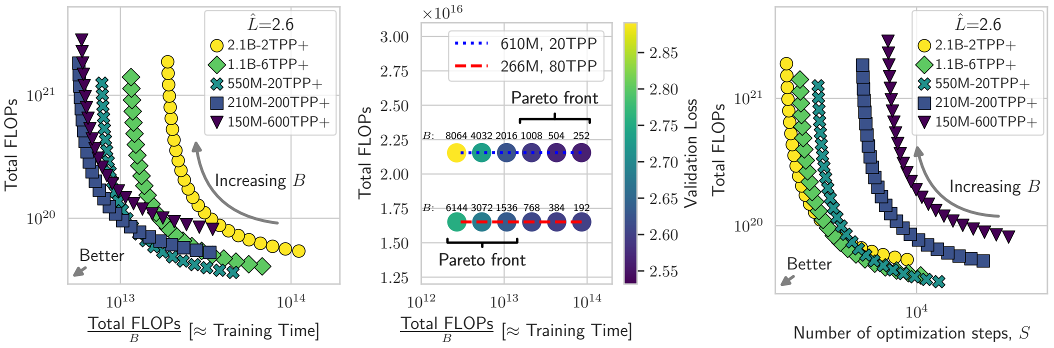


Figure 6: (*left*): Iso-loss curves illustrating time–compute Pareto frontier ($\hat{L}$=2.6). As B increases along curves, more compute (y-axis), but less time (x-axis) is required. Here time $\propto$ $^{\text{Total FLOPs}}/_{B}$. (*middle*): Observed runs where some overtrained models (red line) are on frontier: L in color, B labeled. (*right*): Iso-loss curves, but where time = steps; a very different frontier emerges.

Specifically, when using Training Time $\propto$ $^{\text{Total FLOPs}}/_{B}$, we find overtrained models are FLOP-optimal at certain time budgets (Fig. 6, *left*). Compute-efficient 20 TPP are optimal in pure FLOPs (i.e., ignoring time), as expected, while compute-efficient and overtrained models dominate undertrained ($<$ 20 TPP) models in time and FLOPs. Indeed, this is expected from Eq. (10): *under*training reduces both the excess FLOPs and B_{crit} terms, and thus is never optimal with this model of time.

Finding 8*: When using $B \gg B_{opt}$, it is Pareto-inefficient to train to 20 TPP.*

Notice that Fig. 6 adds "*. . . TPP+*" to curve labels. Here the + sign is a reminder that as we increase B, we require $(1 + B/B_{\text{crit}})$ *extra data* to reach the same $\hat{L}$; i.e., points with higher B are trained to a higher *actual TPP* than the base setting. For example, once the 2TPP+ curve in Fig. 6 reaches 10× its minimum FLOPs, it is *actually* training at 20 TPP. Since starting from an undertrained base setting is never Pareto optimal (as just discussed above), *it is* ***always*** *suboptimal to train a model with a large B to 20 actual TPP*. If large-batch training is needed, the configuration should start from a 20 TPP+ base setting and scale B from there (to $>$20 TPP).

We can see this finding play out in real training runs. Fig. 6 (middle) demonstrates observed runs where our 266M 80TPP models *dominate* our 610M 20TPP models (i.e., in FLOPs *and* time)—when both train with large B. (Note in this plot, results are not iso-loss: the frontier is over L, C, and time.)

***Finding 9**: The Pareto-optimal settings depend on the formulation of time/parallelism strategy.*

While $^{\text{FLOPs}}/_B$ is a good model of data parallel training, it does not incorporate the potential for model parallelism [44]. In the extreme we could assume that all $6N$ FLOPs could be executed concurrently per input token. Under this formulation, training time is proportional only to the number of steps, regardless of model scale. Fig. 6 (*right*) shows the Pareto frontier that would result from this formulation; if we pay no time cost for larger models, we can train faster by using undertrained large models, although, exactly as with overtrained models, they suffer in FLOPs.

While LLMs cannot be fully parallelized due to the inherent sequential nature of a Transformer's layer-by-layer computation, this formulation could be refined by incorporating depth or other architectural features. For example, Inbar and Sernau [45] predict training time via linear regression over total FLOPs and memory-copy operations, fit to real (single-TPU) runs. As formulations improve, different Pareto-optimal configurations will emerge.

***Finding 10**: Inaccurate B_{crit} scaling leads to inaccurate Pareto-optimal configurations.*

Because the Pareto frontier in Sec. 4 depends directly on the B_{crit} power-law fit, any error in that fit produces corresponding errors in the predicted trade-offs between training time and compute. Accurate B_{crit} estimation, in turn, depends on effective λ tuning (Finding 6 and Appendix F.5). When λ is fixed as in standard practice (and thus the $B_{\text{crit}}(D)$ slope is misestimated), B_{crit} will be systematically over- or underpredicted at scale, altering the computed frontier and the apparent Pareto-optimal configurations. To illustrate, artificially varying the B_{crit} exponent changes which models appear on the frontier: as the exponent increases (and B_{crit} rises), higher-TPP models move to the frontier; when B_{crit} is underestimated, only low-TPP (e.g., 20 TPP) models appear Pareto-optimal. Hence, an inaccurate B_{crit} scaling law produces misleading frontiers and can lead to unexpectedly-longer training durations and suboptimal compute allocations.

Recent work suggests that other estimators of B_{crit} (such as those based on the gradient noise scale [11]) can also be systematically biased (Appendix D.2), leading to similar distortions in the Pareto frontier.

***Key takeaway 3**: To balance time and compute at a target loss, select $(N, D_{\min})$ from Eq. (8), determine $B_{crit}(D_{\min})$ via Eq. (7), and use Eq. (9) to estimate compute for any B. Under Time $\propto$ FLOPs$/B$, the resulting time–compute trade-off favors higher D (overtraining) (Fig. 6).*

5 Conclusion

We have presented a comprehensive empirical study of hyperparameter scaling laws in LLM pre-training, focusing on weight decay and batch size. Our approach leverages the AdamW timescale (τ) to develop robust scaling relationships that predict optimal hyperparameter settings across a broad spectrum of model (N), dataset (D), and batch sizes (B). We demonstrated that optimal τ decreases as a power law with the tokens-per-parameter ratio, providing a systematic method to set weight decay optimally across diverse training scenarios.

Furthermore, we introduced a novel, practical methodology for estimating critical batch size (B_{crit}). Our findings diverge from influential prior work that tied B_{crit} predominantly to compute or loss, while agreeing with the recent findings of [12] that underscore dataset size as the principal scaling factor. Additionally, we showed that contrary to previous studies suggesting optimal batch size (B_{opt}) scales primarily with compute, it also exhibits a clear power-law dependence on D.

Also, our analysis of Pareto-optimal configurations reveals an important strategic advantage for smaller, overtrained models in scenarios where rapid training and high parallelism are prioritized.

Appendix B notes limitations and directions for further study suggested by our results. In particular, as inference-time scaling comes to the fore, inference time and compute must also be considered as first-class Pareto objectives. Moreover, finer-grained configuration decisions, such as model depth and context length, should be considered along with N, D, and B.

Acknowledgments and Disclosure of Funding

We thank the NeurIPS reviewers for their helpful feedback. None of the authors received third-party funding or third-party support for this work. None of the authors have financial relationships with outside parties that could potentially be perceived to influence this research.

References

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

[2] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017.

[3] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35, 2022.

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

[5] Greg Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, 2021.

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

[7] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[8] Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020.

[9] Tamay Besiroglu, Ege Erdil, Matthew Barnett, and Josh You. Chinchilla scaling: A replication attempt. *arXiv preprint arXiv:2404.10102*, 2024.

[10] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models, 2023.

[11] Sam McCandlish, Jared Kaplan, Dario Amodei, et al. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[12] Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How does critical batch size scale in pre-training? *arXiv preprint arXiv:2410.21676*, 2024.

[13] Nolan Dey, Gurpreet Gosal, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-GPT: Open compute-optimal language models trained on the Cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023.

[14] Nolan Dey, Daria Soboleva, Faisal Al-Khateeb, Bowen Yang, Ribhu Pathria, Hemant Khachane, Shaheer Muhammad, Zhiming, Chen, Robert Myers, Jacob Robert Steeves, Natalia Vassilieva, Marvin Tom, and Joel Hestness. BTLM-3B-8K: 7B parameter performance in a 3B parameter model, 2023.

[15] Neha Sengupta, Sunil Kumar Sahu, Bokang Jia, Satheesh Katipomu, Haonan Li, Fajri Koto, William Marshall, Gurpreet Gosal, Cynthia Liu, Zhiming Chen, et al. Jais and Jais-chat: Arabic-centric foundation and instruction-tuned open generative large language models. *arXiv preprint arXiv:2308.16149*, 2023.

[16] Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D Cox, and Rameswar Panda. Power scheduler: A batch size and token number agnostic learning rate scheduler. *arXiv preprint arXiv:2408.13359*, 2024.

[17] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[18] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

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

[20] Lucas Lingle. A large-scale exploration of μ-transfer. *arXiv preprint arXiv:2404.05728*, 2024.

[21] Lorenzo Noci, Alexandru Meterez, Thomas Hofmann, and Antonio Orvieto. Why do learning rates transfer? Reconciling optimization and scaling limits for deep learning. *arXiv preprint arXiv:2402.17457*, 2024.

[22] Johan Bjorck, Alon Benhaim, Vishrav Chaudhary, Furu Wei, and Xia Song. Scaling optimal LR across token horizons. *arXiv preprint arXiv:2409.19913*, 2024.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

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

[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[26] Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.

[27] Noam Shazeer. GLU variants improve transformer, 2020.

[28] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. Web page, 2023.

[29] Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Straight to zero: Why linearly decaying the learning rate to zero works best for LLMs. *arXiv preprint arXiv:2502.15938*, 2025.

[30] Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, et al. Surge phenomenon in optimal learning rate and batch size scaling. *arXiv preprint arXiv:2405.14578*, 2024.

[31] Oleg Filatov, Jan Ebert, Jiangtao Wang, and Stefan Kesselheim. Time transfer: On optimal learning rate and batch size in the infinite data limit. *arXiv preprint arXiv:2410.05838*, 2024.

[32] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[33] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The Falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[34] Aleph Alpha. Introducing Pharia-1-LLM: Transparent and compliant. Web page, 2024. Accessed: 2024-09-07.

[35] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *arXiv preprint arXiv:2406.19146*, 2024.

[36] Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, David Zhang, et al. CWM: An open-weights llm for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.

[37] Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable scale: Part I, step law – optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv:2503.04715*, 2025.

[38] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[39] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

[40] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

[41] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *arXiv preprint arXiv:2405.18392*, 2024.

[42] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

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

[44] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[45] Itay Inbar and Luke Sernau. Time matters: Scaling laws for any budget. *arXiv preprint arXiv:2406.18922*, 2024.

[46] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[47] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021.

[48] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*, 2019.

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

[50] Kimi Team. Kimi K2: Open agentic intelligence. Tech report, 2025.

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

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

[53] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024.

[54] Tanishq Kumar, Zachary Ankner, Benjamin F Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *arXiv preprint arXiv:2411.04330*, 2024.

[55] Martin Marek, Sanae Lotfi, Aditya Somasundaram, Andrew Gordon Wilson, and Micah Goldblum. Small batch size training for language models: When vanilla SGD works, and why gradient accumulation is wasteful. *arXiv preprint arXiv:2507.07101*, 2025.

[56] Alexandru Meterez, Depen Morwani, Jingfeng Wu, Costin-Andrei Oncescu, Cengiz Pehlevan, and Sham Kakade. Seesaw: Accelerating training by balancing learning rate and batch size scheduling. *arXiv preprint arXiv:2510.14717*, 2025.

[57] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training Gopher, 2022.

[58] Atli Kosson, Bettina Messmer, and Martin Jaggi. Analyzing & reducing the need for learning rate warmup in GPT training. *arXiv preprint arXiv:2410.23922*, 2024.

[59] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

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

[61] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[62] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

[63] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in neural information processing systems*, 32, 2019.

[64] William Merrill, Shane Arora, Dirk Groeneveld, and Hannaneh Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. *arXiv preprint arXiv:2505.23971*, 2025.

[65] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[66] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016.

[67] Samuel L Smith and Quoc V Le. A Bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017.

[68] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[69] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[70] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the SDEs and scaling rules for adaptive gradient algorithms. *Advances in Neural Information Processing Systems*, 35:7697–7711, 2022.

[71] Twan Van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

[72] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[73] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018.

[74] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[75] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

[76] Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics: Learning dynamics of neural network with normalization, weight decay, and sgd. *arXiv preprint arXiv:2006.08419*, 2020.

[77] Atli Kosson, Bettina Messmer, and Martin Jaggi. Rotational equilibrium: How weight decay balances learning across neural networks. *arXiv preprint arXiv:2305.17212*, 2023.

[78] Francesco D'Angelo, Maksym Andriushchenko, Aditya Vardhan Varre, and Nicolas Flammarion. Why do we need weight decay in modern deep learning? *Advances in Neural Information Processing Systems*, 37:23191–23223, 2024.

[79] Nolan Dey, Quentin Anthony, and Joel Hestness. The practitioner's guide to the maximal update parameterization. Blog post, September 2024.

[80] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

[81] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. *arXiv preprint arXiv:2403.08540*, 2024.

A Broader impacts

This paper presents methods to train LLMs more efficiently: practitioners can use our methods to reduce the total compute FLOPs used to train models, subject to time constraints. Given the intense pressure to advance LLM capabilities as quickly as possible, our methods can therefore reduce the associated environmental and financial costs of LLM training [46, 47].

Moreover, hyperparameter tuning is a key contributor to these costs, and impairs equity in AI research, as tuning success depends directly on researcher finances [48]. We hope our exploration of optimal hyperparameter scaling can reduce the burden of hyperparameter tuning at scale and thus improve equity in AI.

B Limitations

While our findings corroborate prior work and provide strong evidence for the proposed scaling laws in τ, B_{opt}, and B_{crit}, there are several limitations that merit further study.

EMA perspective As the EMA perspective regards parameters y_t as a function of updates x_t, it fails to account for x_t actually depending on earlier values of y_t (e.g., y_{t-1}). Yet although this perspective has formal limitations, we nevertheless find it a useful *conceptual* model of training, as it predicts behavior that is supported by experiments.

Optimization and training setup Our work focuses on AdamW (the standard optimizer for LLM training). While the EMA perspective applies directly to other optimizers that use decoupled weight decay, such as Sophia [49] and MuonClip [50], it may not apply to approximate second order methods, e.g., Shampoo [51]. However, it can be used when applying AdamW (and related optimizers) in Shampoo's eigenbasis, which was shown to be effective in SOAP [52].

We present results with a single (standard) learning rate schedule. Our method for obtaining B_{crit} estimates would be quite efficient with a warmup-stable-decay (WSD) schedule [17, 53], as we could perform a single training run with each batch size, but decay at various milestones in order to get points along the scaling law, essentially following the approach in Hägele et al. [41], but with separate laws for each batch size.

We used the maximal update parameterization in all experiments, which generates a learning rate η adjustment for each model width. Our results suggest this approach enables good models at arbitrary N, D, and B when combined with adjustments to λ. This strategy is informed by our experiments comparing re-adjusting η vs. λ in Sec. 2.4. However, it is not feasible, at this scale, to verify whether substantially better models could be obtained by sweeping the full cross-product of η and λ values.

Our study specifically focuses on the practically important setting of single-epoch LLM pre-training. Wang and Aitchison [1] indeed noted differences in optimal τ when using multi-epoch training, possibly due to data repetition. Reconciling these differences by isolating the effects of repetition versus scale is an interesting follow-up direction.

Here we only experimented with a single dataset, vocabulary, and context length. We obtained a similar B_{crit} scaling law to Zhang et al. [12], but it would be interesting to see if differences in the coefficient of our power laws could be attributed to specific differences in approach (e.g., differences in dataset, context length, learning rate schedule, use of weight decay, etc.). Appendix D.3 has further discussion of differences with Zhang et al. [12].

We have also not explored how changes in numerical precision could affect scaling laws. Recent work [54] showed that, in terms of scaling laws, lower precision reduces the model's *effective* parameter count. This suggests precision would have no impact on scaling of B_{opt} or B_{crit}, which do not scale in N. Lower precision, however, could increase the *effective* TPP (via smaller effective N), thereby altering τ_{opt}.

Small batches, large batches, and dynamic batch sizing We consistently find that smaller and smaller batches do not grow asymptotically closer to D_{min}, as predicted by theory, but eventually degrade in loss. One possibility is that λ tuning is not sufficient with very small B, and further tuning of other hyperparameters may be needed, such as the Adam β parameters (as suggested in recent

Table 2: Model architectures used in experiments

Model	d_{model}	n_{layers}	d_{head}
111M	768	10	64
266M	768	32	64
610M	2048	10	64
1.7B	2048	32	64
3.3B	2048	64	64

Table 3: Models, tokens-per-parameter (TPP) and corresponding dataset sizes (in tokens) used in main experiments. We also list the total number of batch sizes, B, trained at each scale and TPP, as well as the number of B for which we tuned λ. For B where λ was not tuned, it was inferred via the τ_{opt} scaling law (Sec. 2). Additional sweeps of η were done at each B at 610M-20TPP scale for the experiments in Sec. 2.4. Around 400 different LLMs were trained in total across all the experiments.

Model	TPP	D	Number of B	Number of B with λ tuned
111M	20	2.19B	8	8
111M	80	8.76B	8	8
111M	200	21.9B	7	7
111M	320	35.0B	8	8
111M	1280	140.1B	6	1
266M	20	5.31B	7	7
266M	80	21.2B	7	7
266M	320	85.0B	6	6
266M	1280	339.8B	1	0
610M	20	12.1B	8	8
610M	80	48.5B	7	7
610M	200	121.3B	6	6
610M	320	194.1B	2	1
1.7B	20	34.3B	7	7
1.7B	80	137.2B	7	1
1.7B	320	548.6B	1	0
3.3B	20	66.5B	1	0
3.3B	23	76.5B	1	0
3.3B	30	99.8B	2	1

work [35, 12, 55]). Some preliminary tests using the β_2 scaling rule from Marek et al. [55] showed loss improvements at small B. Since we are unlikely to train with small batches at scale, and using them even with smaller LLMs significantly impairs our ability to train both efficiently and quickly, it is unfortunately difficult to justify further exploration in this direction.

Regarding large batches, our methods do not account for the many practical systems-related issues, including bandwidth and communication overheads, memory limits of hardware, synchronization delays, etc. Moreover, as batch sizes increase, techniques such as optimizer sharding may be needed, which further complicate performance model [33]. Our scaling laws do, however, explicitly define a practically relevant regime of training batch sizes: $B_{\text{opt}} \leq B \leq B_{\text{crit}}$. Practitioners can leverage this identified regime alongside system-specific profiling (e.g., evaluating utilization at various batch sizes) to select optimal settings balancing algorithmic and systems constraints.

Exploring optimal dynamic batch sizing is a natural future direction for our work. While the potential gains were found to be small in theory by McCandlish et al. [11], more recent work has found significant wall clock speedups [56].

C Experimental Details

Table 2 provides details on the model architecture and hyperparameters for models used in the experiments. Table 3 provides, for each model scale and TPP, the dataset sizes used in training, the

Table 4: Tuned hyperparameters for μP proxy model

$\sigma_{W,\text{base}}$	8.67e-02
$\tilde{\eta}$	1.62e-02
α_{input}	9.17
α_{output}	1.095

number of batch sizes tested, and the number of batch sizes for which λ was tuned. Around 400 models in total were trained for the main experiments.

All the models in our main experiments were trained on the SlimPajama dataset [28], a cleaned and deduplicated version of the RedPajama dataset. We use the GPT-2 [25] vocabulary of size 50257, and a context length of 2048 tokens. Following standard practice, we do not apply weight decay or bias to LayerNorm layers. AdamW settings are $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 1\text{e}{-8}$. Validation loss is always computed over a held-out 1.1B tokens, regardless of training TPP. We report cross-entropy loss. By default we parameterize with μP, with hyperparameters set via proxy tuning, as described below.

For a given TPP, all models have the exact same warmup phase: a linear warmup of the learning rate from 0 to the maximum value. In all our runs, warmup was 10% of the total steps. Learning rate warmup is standard practice in LLM pre-training [32, 57, 42, 43, 58].

All models in the main experiments were trained on a Cerebras CS-3 system. 610M-parameter 20TPP models take roughly 6 hours each to train on a single CS-3.

For a given model configuration, we find results to be very stable across random seeds. To quantify the variance, we repeated 111M-parameter, 20 TPP training four additional times for six different hyperparameter settings, resulting in 5 total validation loss results for each of the six training runs. Standard deviation of the validation loss was below 0.003 in all cases.

Proxy model hyperparameter tuning To find the optimal μP hyperparameters (HPs), we trained a 39M proxy model using a width d_{model} of 256, with 24 layers and head size of 64. We trained this model on 800M tokens with a batch size of 256 sequences and a context length 2048. We randomly sampled 350 configurations of base learning rates, base initialization standard deviation, and embedding and output logits scaling factors, and used the top-performing values as our tuned HPs (Table 4).

D Additional related work

D.1 Optimizers for large-batch training

Prior work has explored optimizers designed specifically for large-batch training, including LARS [59] and LAMB [60]. It is instructive to consider these prior findings in light of the scaling laws from our paper. In particular, both original BERT [61] and the LAMB replication were trained on 85.2B tokens. Applying our fitted B_{crit} power law over D=85.2B, we obtain an estimated B_{crit} of about 12M tokens. Original BERT was trained for 90% of steps with a batch size of 65K tokens (512 sequences of length 128). LAMB increased the batch size to 4M tokens (32K sequences), justifying their claim, "BERT training can be reduced from 3 days to just 76 minutes" [60]. However, based on the predicted B_{crit} of 12M, batch size 4M is still well within the expected range of efficient batch sizes. Moreover, the LAMB paper later notes, "we did not observe any speedup by increasing the batch size from 65536 to 131072 [sequences, or 16.8M tokens]." In other words, they reach the point of diminishing return *exactly where B exceeds our predicted B_{crit}.*

It is likely that some optimization issues solved by LAMB (to enable stable large-batch training) are solved other ways in modern LLM training setups, via, e.g., gradient clipping, pre-LayerNorm placement, better initialization and stability control through μP, etc. Scaling λ rather than η with B, as we propose, further supports stable, efficient training. However, gradient redundancy imposes an inherent limit on useful batch sizes, ensuring critical batch size remains relevant.

D.2 Critical batch size

Observations of critical batch size have previously been related to data complexity [62], loss curvature [39, 63], and model architecture [40].

Merrill et al. [64] define B_{crit} as the largest B such that loss does not degrade by more than a fixed fraction ϵ from the B_{opt} setting. They measure this B_{crit} instantaneously throughout training, by repeatedly branching from a checkpoint with different B settings and assessing the impact on loss.

Recent work also defines B_{crit} in terms of how η scales with B [31, 30]; unlike our work, these recent studies use a constant learning rate schedule and no weight decay.

We follow McCandlish et al. [11]'s definition of B_{crit} (Definition 3.1). Given various theoretical assumptions, McCandlish et al. [11] derived a direct equivalence between B_{crit} and what they call the *gradient noise scale* (GNS): the variation of the gradients between different training examples. However, they noted that the GNS "accurately predicts the largest usable batch size (at the order of magnitude level)," which is below the level of precision needed for large-scale training. Merrill et al. [64] recently found "the gradient noise scale underestimates the CBS [i.e., B_{crit}]." This lack of precision may be why, in Kaplan et al's original scaling laws paper [4], they note that, "although the critical batch size roughly matches the gradient noise scale, we are using a direct [empirical] measurement of B_{crit}." Our approach to measuring B_{crit} (Sec. 3.2) similarly provides a direct empirical measurement, but one that can be efficiently computed with any learning rate schedule or optimizer.

D.3 Detailed comparison with Zhang et al. [12]

Here we provide further comparison with the concurrent work by Zhang et al. [12]. The primary point of distinction of our paper is that we conducted a large-scale empirical study into the scaling of AdamW's weight decay hyperparameter (including its scaling with B), ultimately deriving a precise power law for the optimal AdamW timescale in tokens-per-parameter. Zhang et al. [12] did not use weight decay. Further, we also explored scaling of B_{opt} in addition to B_{crit}. Beyond use of weight decay, further methodological differences in our main experiments include that we used a longer context length (2048 vs. 512), a cleaner dataset (SlimPajama vs. C4), the μP parameterization, a decaying LR schedule (more on this below), and that we tuned HPs at most N, D, B (Table 3), while [12] performed a HP sweep for a 151M model, and re-used optimal values at other scales. We now focus on differences in estimating and measuring the scaling of B_{crit}.

Estimating B_{crit} for a specific target loss Both our work and Zhang et al. [12] require measuring, for different batch sizes, how many training steps it takes to reach a particular target loss. Since the number of steps to reach that loss is not known *a priori*, it is inherently difficult to study B_{crit} when using a LR decay schedule, where you must specify the number of steps in advance. Using a constant LR (as was done in early work on B_{crit} [11]) simply does not result in competitive models [29]. Unfortunately, it is not feasible to *search* for the precise step count needed, i.e., by conducting full training runs with different schedules/step budgets.

Zhang et al. [12] creatively solve this issue by conducting a single training run at a constant LR, while using weight averaging to generate higher-quality checkpoints for evaluation. With this approach, they still "need to frequently evaluate the model on a holdout evaluation set" [12].

Given LR decay, as opposed to weight averaging, remains the standard practice for current state-of-the-art LLMs, we independently developed a different approach. This led to the novel method described in our paper. In contrast with [12], we do not need to frequently evaluate the model, as we instead fit a B-specific loss power law through a few validation loss values (Sec. 3.2). Indeed, our approach may improve the efficiency of [12]'s method, as it would obviate the cost of continuous validation, which concerned them (see their section "Evaluation data size and frequency").

Estimating the B_{crit} power law Collecting B_{crit} data across multiple model scales and loss targets is expensive. Zhang et al. [12] establish B_{crit} scaling in D through three targeted experiments:

- measuring B_{crit} while scaling N but leaving D fixed to 3.07B
- measuring B_{crit} while scaling D but leaving N fixed to 302M
- measuring B_{crit} while scaling both N and D proportionally (at 20 TPP)

Interestingly, B_{crit} was found to only scale weakly in N, but scale similarly whenever D is scaled. They then fit a power law to their data points for the 302M-parameter model, obtaining the fit $B_{\text{crit}} = 22.91D^{0.47}$ (in tokens).

In comparison, we took a more brute-force approach, computing B_{crit} across many different N and D values, and ultimately fitting our B_{crit} power law across multiple different model sizes and TPP settings (Fig. 1, *right*). Also, unlike [12], we assessed the quality of fit via computation of both R^2 values and parameter quantiles via bootstrapping (Sec. 3.3).

Recall also that [12] used a different definition of critical batch size. Let us denote their quantity B_{zhang}. They set B_{zhang} to be the B such that the data required to reach a loss target is $1.2 \times D_{\text{min}}$ (i.e., $1.2\times$ the data required with B_{opt}).

We can use Eq. (6) to align their fitted law with our own. By this equation, we have:

$$\begin{aligned} D &= D_{\text{min}}(1 + \frac{B_{\text{zhang}}}{B_{\text{crit}}}) \\ &:= D_{\text{min}}(1.2) \\ \Rightarrow \frac{B_{\text{zhang}}}{B_{\text{crit}}} &= 0.2 \\ \Rightarrow B_{\text{crit}} &= 5B_{\text{zhang}} \end{aligned}$$

Thus, to convert their coefficient to our scale, we multiply it by 5, and, dividing by the number of tokens in our sequences, obtain $B_{\text{zhang}} = 0.0559D^{0.47}$. The B_{zhang} coefficient (0.0559) is 19% larger than our own (0.0471), perhaps reflecting differences in training setup or data quality (and worth investigating further in future work). However, the exponents are quite similar (0.47 vs. 0.462), suggesting that both works are independently measuring the same fundamental scaling behavior.

We emphasize that B_{crit} directly reflects the fundamental limit to data parallelism in training neural networks. Given the significant implications of B_{crit} scaling in D rather than C or L (including those discussed in Sec. 4), we note the scientific and practical value in having different approaches independently observe this same phenomenon.

D.4 Hyperparameter scaling with B

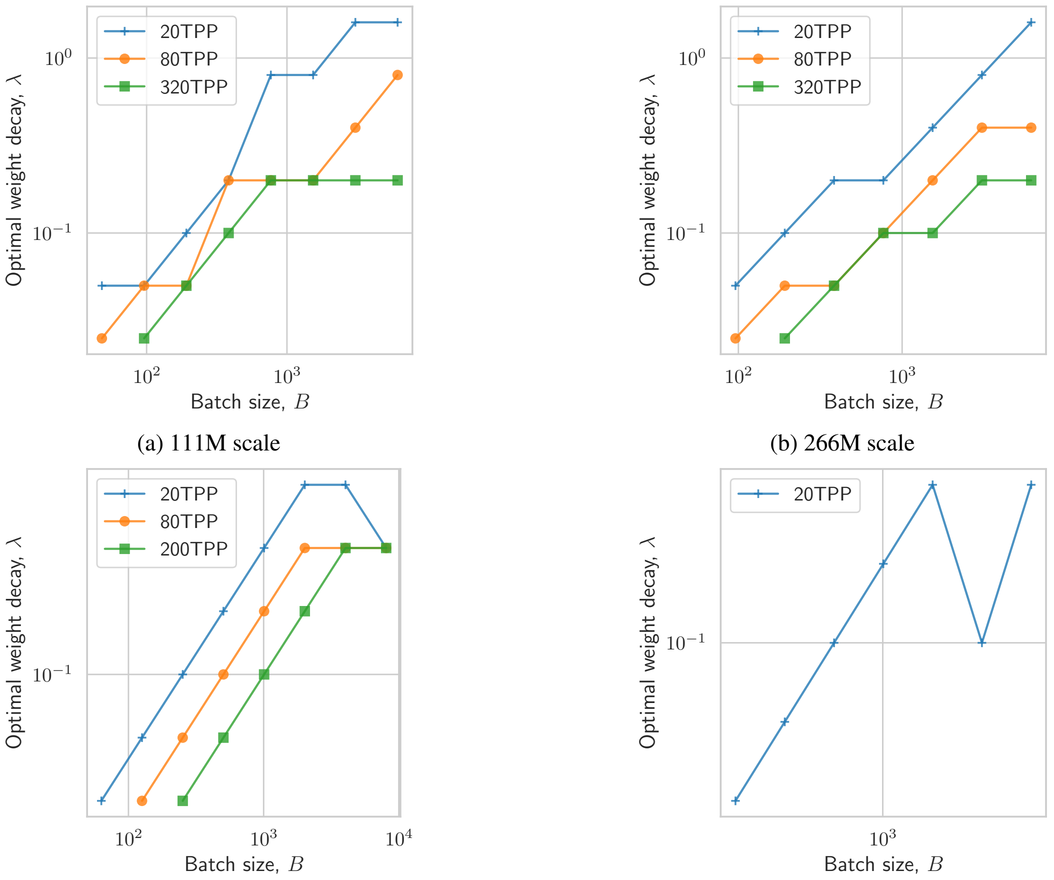
It has long been recognized that the optimal learning rate, η_{opt}, scales with B, with reports of both linear [65–68] and square-root scaling [69, 60, 70]. Recent work has found η_{opt} to *decrease* when $B > B_{\text{crit}}$ [30, 31], which resonates with our own findings (Fig. 2, *right*). Since it is difficult to predict exactly how η will scale with B, studies of B_{crit} have often done full HP sweeps at each B [11, 40].

The only work we are aware of that specifically recommends scaling weight decay with B is Loshchilov and Hutter [23], who suggest $\lambda \propto \sqrt{B}$, though this rule is not evaluated systematically. It is also important to note that Loshchilov and Hutter [23] use the *independent* form of weight decay, where decay is applied independently of the learning rate η, unlike common implementations such as AdamW in PyTorch [6]. In these more typical *dependent* implementations, weight decay is scaled by η, so any increase in η with B (e.g., $\eta \propto B$ or $\sqrt{B}$) already increases the effective weight decay strength accordingly.

D.5 τ and effective learning rates

The concept of effective learning rates, influenced by weight decay, has been widely discussed [71–78]. In its simplest form, the effective or *intrinsic* LR is simply $\eta\lambda$, but in these prior works, effective LRs typically measure functional updates relative to weight magnitude, which is particularly relevant for normalization-based networks. Comparison of the effects of λ vs. η adjustments in the context of LR decay schedules was explored in [29].

The behavior of effective LRs (relative update sizes) over the course of training has been studied comprehensively by Kosson et al. [77], including comparing the effects of increasing η vs. increasing λ. This work shows that higher η values can cause large relative updates early in training, which can destabilize training or require longer warmups [58]. High η and low λ can also lead to larger weight norms [77, 78], which also has a destabilizing effect, particularly on low-precision training. These



(a) 111M scale

(b) 266M scale

(c) 610M scale

(d) 1.7B scale

Figure 7: **Optimal weight decay scaling with** B: The optimal weight decay increases linearly over small batch sizes—until $B > B_{\text{crit}}$.

effects may explain why we were able to achieve higher effective LRs $\eta\lambda$ by tuning λ rather than tuning η with B (Fig. 2, *middle*).

For a given dataset size D, the τ and the batch-normalized effective LR $\frac{\eta\lambda}{B}$ are equivalent, and thus effective LRs and the AdamW timescale can be viewed as different perspectives on the AdamW optimization process.

E Scaling of τ and λ: additional details and results

E.1 λ scaling with B

Fig. 7 shows how optimal λ changes across B, for all of the model scales and TPP levels where we did hyperparameter sweeps. There is a strong linear relationship between λ and B over the smaller batch sizes $B < B_{\text{crit}}$, with optimal λ eventually plateauing (or decreasing). Note the standard use of λ=0.1 [3, 32–34] is only optimal at specific B, and this B changes with TPP.

E.2 Additional details on τ fitting

We now describe how we obtained the optimal τ values at specific model scales and TPP ratios. Rather than taking the empirical minimum loss, we fit a parabola to the $\langle L, \tau \rangle$ points in log space and took the analytic minimum of the parabola. If we have multiple loss values at the same τ (e.g., our data for a single scale and TPP comprises multiple different batch sizes), we only kept the lowest loss points at each τ prior to parabola fitting. We used validation loss on the held-out validation set. For

Algorithm 1 Generating the optimal τ power law

```
Input: small batch size $B$, optimal per-$N$ learning rates $\eta$
Initialize $tau\_scaling\_law\_fitting\_points = [\,]$
for $N$ in $model\_scales$ do
  for $D$ in $10N, 20N, 80N, 320N, \ldots$ do
    Reset $loss\_points = [\,]$
    for $\lambda$ in $lambda\_range$ do
      Train $LLM(N, D, B, \lambda, \eta)$, get validation loss $L'$
      $\tau = {}^{B}/_{\eta\lambda D}$
      $loss\_points[\tau] = L'$
    end for
    $\tau_{\text{opt}} = \arg\min_{\tau}(loss\_points)$
    $tau\_scaling\_law\_fitting\_points.add(\langle \text{TPP}={}^{D}/_{N}, \tau_{\text{opt}}\rangle)$
  end for
end for
Fit $c$, $m$ for $\tau_{\text{opt}} = c\text{TPP}^{m}$ on $tau\_scaling\_law\_fitting\_points$
```

our τ calculations, we input B in units of *tokens* (in contrast to the *output* of our reported B_{opt} and B_{crit} scaling laws, which we report in units of sequences, of 2048 tokens). Algorithm 1 sketches the full procedure for generating the τ power law (Eq. (3)).

E.3 Relationship to prior power laws

E.3.1 Relationship to η_{opt} scaling laws in dataset size, D

Both Shen et al. [16] and Bjorck et al. [22] propose scaling laws for the optimal learning rate, η_{opt}, as a power law in the amount of data, D:

$$\eta_{\text{opt}} = B \cdot c_{\eta_D} \cdot D^{m_{\eta_D}}$$

We now discuss how this power law also follows from the power law of τ_{opt} in TPP. By Eq. (3), we have:

$$\begin{aligned}\tau_{\text{opt}}(\text{TPP}) &= c_{\tau} \cdot \text{TPP}^{m_{\tau}} \\ &= c_{\tau} \cdot \left(\frac{D}{N}\right)^{m_{\tau}}\end{aligned}$$

Substituting in the definition of τ (Eq. (2)), and assuming λ, B, and N are fixed,[3] this implies η_{opt} will scale as:

$$\begin{aligned}\frac{B}{\eta_{\text{opt}}\lambda D} &= c_{\tau} \cdot \frac{D^{m_{\tau}}}{N^{m_{\tau}}} \\ \Rightarrow \eta_{\text{opt}} &= B\left(\frac{N^{m_{\tau}}}{\lambda \cdot c_{\tau}}\right) D^{-(m_{\tau}+1)} \\ &= B \cdot c_{\eta_D} \cdot D^{m_{\eta_D}} \qquad (11)\end{aligned}$$

$$\text{where} \quad c_{\eta_D} = \frac{N^{m_{\tau}}}{\lambda \cdot c_{\tau}} \quad \text{and} \quad m_{\eta_D} = -(m_{\tau}+1)$$

Eq. (11) is exactly the form of the power law used in Shen et al. [16], and explains the results across batch sizes seen in Bjorck et al. [22], as discussed in Sec. 2.4.

Comparison to fit in Power Scheduler [16] Given $c_{\eta_D} = \frac{N^{m_{\tau}}}{\lambda \cdot c_{\tau}}$ and $m_{\eta_D} = -(m_{\tau}+1)$, we can use these formulas to compare our fit coefficients to those in Shen et al. [16].

In Shen et al. [16], they find $m_{\eta_D} = -0.51$. In our case, $m_{\tau} = -0.520$, and therefore $m_{\eta_D} = -0.48$, which is quite similar.

[3] Bjorck et al. [22] use a fixed λ=0.1, a fixed batch size of 0.5M tokens (for most of their experiments), and fit scaling laws separately at different model sizes (except in their Section 4). Shen et al. [16] use μP to adjust the LR for different model scales, so the derivation applies at any particular N; they do not report which optimizer is used nor any of its settings.

Comparing our c_{η_D} to their c_{η_D} (= 4.6) is a bit less straightforward. First of all, Shen et al. [16] inputs B in sequences (of length 4096). We thus convert to the scale of our coefficient by dividing by their sequence length, obtaining $c_{\eta_D} = 0.0011$. Secondly, the power law of Shen et al. [16] is actually for the *base* μP learning rate $\tilde{\eta}$, while our derivation above assumes the adjusted (final) learning rate η (Sec. 2.1).

Let us first compare c_{η_D} coefficients at the proxy-model scale, i.e., where $\eta = \tilde{\eta}$. If we were to use a 28M-parameter proxy model, and a default λ=0.1 (and using our fit values of $c_\tau = 1.084$ and $m_\tau = -0.527$), then, by $c_{\eta_D} = \frac{N^{m_\tau}}{\lambda c_\tau}$, our c_{η_D} would also equal 0.0011.

Now we consider how our coefficient varies when N scales. To convert our η scaling law into one for the base $\tilde{\eta}$, we can instead use $c_{\eta_D} = \frac{N^{m_\tau}}{\lambda \cdot \rho \cdot c_\tau}$, where $\rho = P/W$, P is the width of the proxy model, and W is the width of the target model. The width also affects the number of parameters, N, and hence the term N^{m_τ}. In Transformers, N scales roughly as $N \propto LW^2$, where L is the model depth and W is the model width. If we round the fitted exponent to $m_\tau \approx -0.5$, and substitute the value $\rho \propto 1/W$ into the denominator, we therefore have:

$$\begin{aligned} c_{\eta_D} &= \frac{N^{m_\tau}}{\lambda \rho c_\tau} \\ &\propto \frac{N^{-0.5}}{\frac{1}{W}} \\ &\propto \left(LW^2\right)^{-0.5} W \\ &\propto L^{-0.5} \left(W^2\right)^{-0.5} W \\ &\propto L^{-0.5} \end{aligned}$$

which is invariant to changes in W—i.e., the μP adjustment cancels out the model scaling in width. So, if we only scale W, τ scaling would stay in agreement with the Power Scheduler recipe, but if we increase depth, τ scaling would decrease η proportional to $1/\sqrt{L}$ in a manner that is not accounted for in Shen et al. [16].

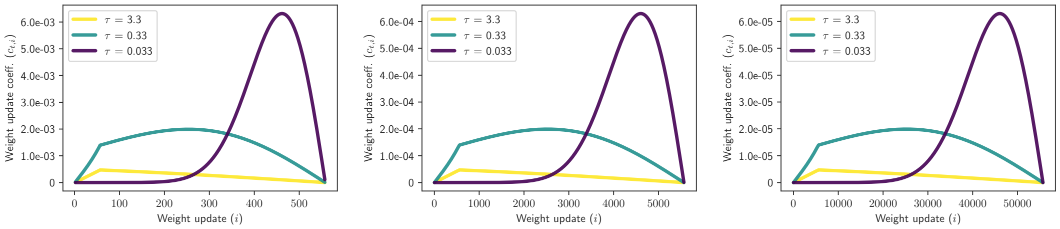
The key point is that the scaling law used by Shen et al. [16] is valid, indeed, has similar fitted exponents, to what would be predicted by the τ_{opt} scaling law—but in a specific context only (small models, or models only scaling in width). Moreover, we have shown it may be less effective to adjust η in order to optimize τ (as these approaches implicitly do); we obtained superior results by instead adjusting λ. By considering η, λ, and B holistically, our scaling laws are a superset of these laws for η_{opt}, as well as other laws that we discuss further presently.

E.3.2 Relationship to η_{opt} scaling laws in model size, N

Sec. 2.2 gave our recipe for tuning hyperparameters, for an arbitrary N, D, and B setting. Here we advocated setting peak η to the μP-adjusted learning rate (where the base learning rate comes from proxy-tuning). Rather than further adjusting this LR based on the dataset size or batch size, we argued for instead adjusting λ so that τ is tuned to its optimal value. Based on both theory, and our empirical findings comparing tuning η to tuning λ, we believe that using μP to scale η_{opt} with model width is sufficient for well-tuned models. That is, the *theoretical* scaling law for η_{opt} (in model width), given by μP, is sufficient for good performance. We discuss this perspective further in this section, specifically how the μP scaling law can explain recent work in *empirical* η_{opt} power laws.

As noted in Sec. 2.1, when using μP, a base η is tuned on a small proxy model, and then scaled depending on the width of the target model. Let W be the width of the target model, and let P be the width of the proxy model. μP prescribes scaling the optimal base learning rate, $\tilde{\eta}_{\text{opt}}$, down to $\eta_{\text{opt}} = \rho\tilde{\eta}_{\text{opt}}$, where $\rho = P/W$. That is, $\eta_{\text{opt}} = P\tilde{\eta}_{\text{opt}}/W$. As models grow in size, P and $\tilde{\eta}_{\text{opt}}$ do not change, so η_{opt} will scale $\propto 1/W$. Dey et al. [79, Figure 2] show that, indeed, a range of LLMs from the GPT, Llama, and DeepMind series are roughly following a scaling law where their chosen learning rate, η is following $\eta \propto 1/W$. In other words, if one were to build a scaling law for η_{opt} based on published LLM settings, it would roughly obey the μP theoretical scaling law.

Furthermore, we can develop a scaling law for η_{opt} in model size, N, using μP, and show that it matches a recent empirical scaling law by Porian et al. [35]. The number of model parameters in any Transformer-based LLM scales roughly in the depth, L, and width, W, as $N \propto LW^2$. If we



(a) 557 steps

(b) 5568 steps

(c) 55680 steps

Figure 8: τ **is invariant to steps, with a learning rate decay schedule** (111M scale, proxy-tuned peak η with linear decay-to-zero): Here we adjust weight decay, λ, in order to maintain τ at a constant value, decreasing λ proportional to the increase in S. Regardless of the total number of optimization steps, we see that the same τ corresponds to the same *shape* of the distribution of weight update coefficients (i.e., the same shape over the *data*, regardless of how the data is discretized in training). For batch sizing, this means that if we use a constant D but increase B by $K\times$ (decreasing S by $K\times$), we will incorporate information across the data similarly—provided we use the same τ.

assume that we maintain a fixed width-to-depth ratio, i.e., $R = W/L$, or $L = W/R$, then we have $N \propto W^3$, or $W \propto N^{1/3}$. Now, since μP prescribe scaling $\eta_{\text{opt}} \propto W^{-1}$, then for a fixed aspect ratio, $\eta_{\text{opt}} \propto N^{-1/3}$.

Taking a very different approach, Porian et al. [35] developed an empirical scaling law for η_{opt} as a function of the number of model parameters. At each model scale, they trained with a variety of batch sizes and learning rates, and found the optimal settings of these hyperparameters. All models were trained to 20 TPP. They then fit a power law through the optimal LR settings, and found that $\eta_{\text{opt}} \propto N^{-1/3}$, exactly as would be expected if one simply followed μP.

As it provides a principled approach to scaling hyperparameters, μP can adapt to scaling when aspect ratio is not fixed. We therefore advocate using μP to set η_{opt}, rather than fitting special η_{opt} power laws. However, with regards to our overall approach, it does not actually matter whether one uses the μP theoretical scaling law *or* an empirical one. The key point is that these laws can be used to set η at a particular model scale, while the τ law should further be used to set λ depending on the B or D values.

E.4 The EMA perspective and learning rate schedules

To understand how the EMA view applies with a dynamic LR schedule, we follow the discussion of Bergsma et al. [29], who extended the formulation in Wang and Aitchison [1]. [29] consider EMAs with time-varying smoothing, $\alpha_t \in [0, 1]$. Letting $\alpha_1 = 1$ (i.e., $y_1 = x_1$), they express y_t in terms of all inputs x_t:

$$
\begin{aligned}
y_1 &= \alpha_1 x_1, \\
y_2 &= (1-\alpha_2)\alpha_1 x_1 + \alpha_2 x_2, \cdots \\
y_t &= \sum_{i=1}^{t} \left(\prod_{j=i+1}^{t} (1-\alpha_j) \right) \alpha_i x_i
\end{aligned}
\tag{12}
$$

The EMA coefficient on each input is denoted $c_{t,i}$, where $c_{t,i} = \left(\prod_{j=i+1}^{t} (1-\alpha_j) \right) \alpha_i$. In other words, $c_{t,i}$ reflects the contribution of input x_i to output y_t at time t, such that $y_t = \sum_{i=1}^{t} c_{t,i} x_i$. Unlike a standard EMA with a fixed smoothing parameter, in this extended EMA the coefficients need not decrease exponentially as i decreases. Indeed, any set of coefficients can be generated by some particular smoothing schedule.

In terms of learning rate schedules for AdamW training, $\alpha_t = \eta_t \lambda$ becomes the smoothing parameter at step t (cf. Sec. 2.1). The EMA itself, y_t, is the model parameters. The EMA is over weight updates: a large coefficient $c_{t,i}$ means the ith weight update contributes a lot to the EMA at step t. The EMA coefficients thus provide a more granular view of the contribution timescale than τ alone.

We now study the question, how do the EMA coefficients change as the step count changes? We generated the $c_{t,i}$ coefficients for the linear decay-to-zero LR schedule, and plot these coefficients at the final step (i.e., showing the contribution of weight updates to the final parameters). We use the μP-tuned and adjusted peak η, for 111M models. The learning rate increases linearly to the peak for the first 10% of steps, then decreases from the peak to 0 for the remainder of steps. We simulated three cases: where we take 557 steps, where we take 5568 steps, and where we take 55680 steps (5568 steps would be 20 TPP for a 111M model using B=192). From the perspective of batch sizing, these different steps could be achieved by decreasing the batch size twice by $10\times$.

We adjusted λ for each step count in order to obtain the same three specific τ values. In Fig. 8, we see that the same τ implies the same *shape* of coefficients across the steps, and hence the same contribution over (normalized) time. That is, weight updates from the same portion of the training data contribute equally to the final model parameters.[4]

The key takeaway is that since τ is independent of the number of steps, it theoretically provides a B-independent measure of the AdamW timescale over weight updates, regardless of learning rate schedule. However, this equivalence for different B breaks down when $B > B_{\text{crit}}$ and weight updates themselves no longer contain linearly $B\times$ the information of a single sample.

F Scaling of B_{opt} and B_{crit}: additional details and results

F.1 Derivation of "extra data" Eq. (6)

Eq. (5) can be written as:

$$\begin{aligned}\frac{D - D_{\text{min}}}{D_{\text{min}}} &= \frac{S_{\text{min}}}{S - S_{\text{min}}}\\ \Rightarrow (D - D_{\text{min}})(S - S_{\text{min}}) &= D_{\text{min}} S_{\text{min}}\\ \Rightarrow DS - DS_{\text{min}} - SD_{\text{min}} &= 0\end{aligned}$$

Given $B = D/S$, we can substitute in $S = D/B$ to get an equation in a single variable, from which we can solve for D.

$$\begin{aligned}\frac{D^2}{B} - DS_{\text{min}} - \frac{DD_{\text{min}}}{B} &= 0\\ \Rightarrow D^2 - DBS_{\text{min}} - DD_{\text{min}} &= 0\\ \Rightarrow D(D - BS_{\text{min}} - D_{\text{min}}) &= 0\\ \Rightarrow D &= D_{\text{min}} + BS_{\text{min}}\end{aligned}$$

Given $B_{\text{crit}} = D_{\text{min}}/S_{\text{min}}$, we can substitute $S_{\text{min}} = D_{\text{min}}/B_{\text{crit}}$ and obtain:

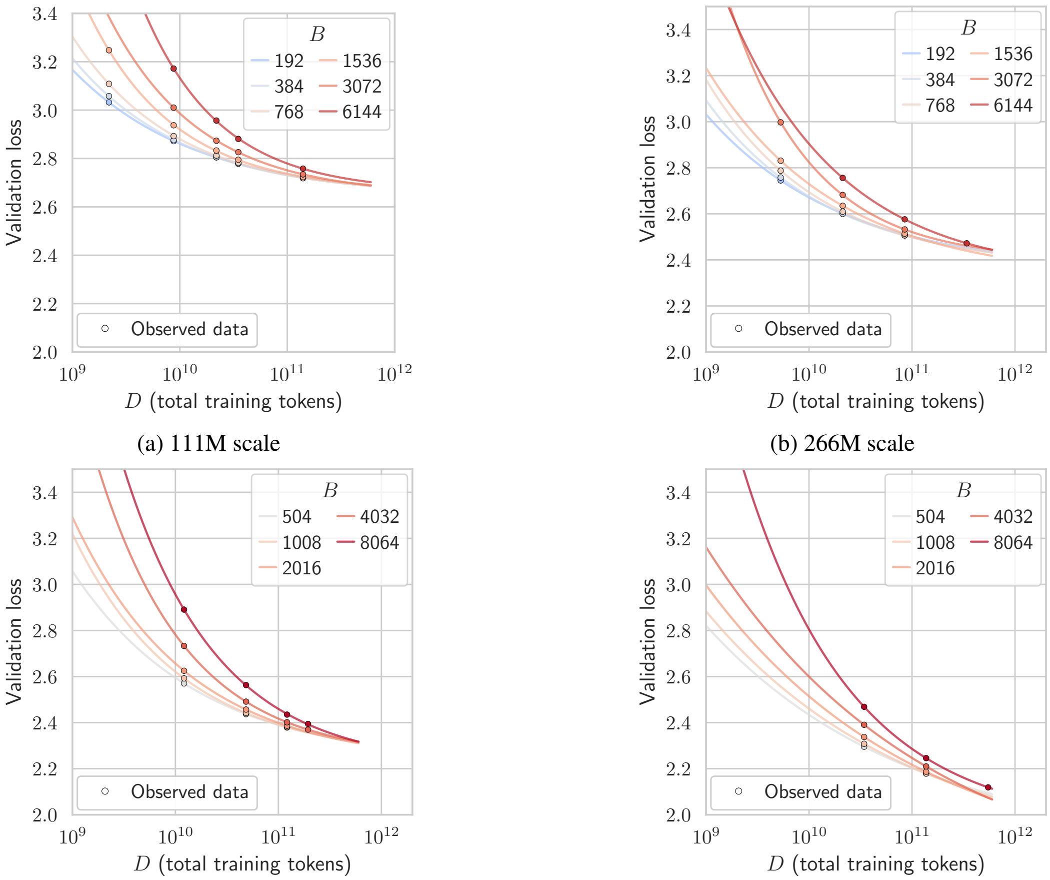
$$\begin{aligned}\Rightarrow D &= D_{\text{min}} + B\frac{D_{\text{min}}}{B_{\text{crit}}}\\ \Rightarrow D &= D_{\text{min}}\left(1 + \frac{B}{B_{\text{crit}}}\right)\end{aligned}$$

F.2 Estimating B_{crit}

We first provide some learnings from developing the B_{crit} estimation procedure.

First, regarding the functional form $L_B(D) = E_N + D_{\text{const}}D^{-\beta}$, we found that including the irreducible loss term E_N was important for obtaining good fits. E_N conceptually represents the Bayes risk *plus* the minimum loss obtainable for a model of size N (i.e., the first two terms of Eq. (8)). Second, only *interpolated* points were reliable; we only compute B_{crit} for loss values where all points are between, or very near to, curve fitting points. Third, each power law should have at least 3 points for fitting, in order to capture the concavity of the scaling in D. Finally, we sample our B values logarithmically and, as in McCandlish et al. [11], perform our fits to Eq. (5) in log space.

[4] Note we do not plot the initial coefficients $c_{t,0}$ here, but they are equal across the scales for a given τ, unlike the non-initial coefficients, which reduce by $10\times$ as the number of steps increases by $10\times$. So a constant τ means both the same contribution across the data, *and* the same *bias* (dependence on initial weights).



(a) 111M scale

(b) 266M scale

(c) 610M scale

(d) 1.7B scale

Figure 9: **Scaling laws in D for computing B_{crit}**: Full set of B-specific power laws, for 111M to 1.7B scales, fitted after training models with different batch sizes, B, and dataset sizes, D, at each scale (empirical data from real training runs indicated by points). From these power laws, we can compute the amount of data needed to reach any target loss, as illustrated in main paper Fig. 3.

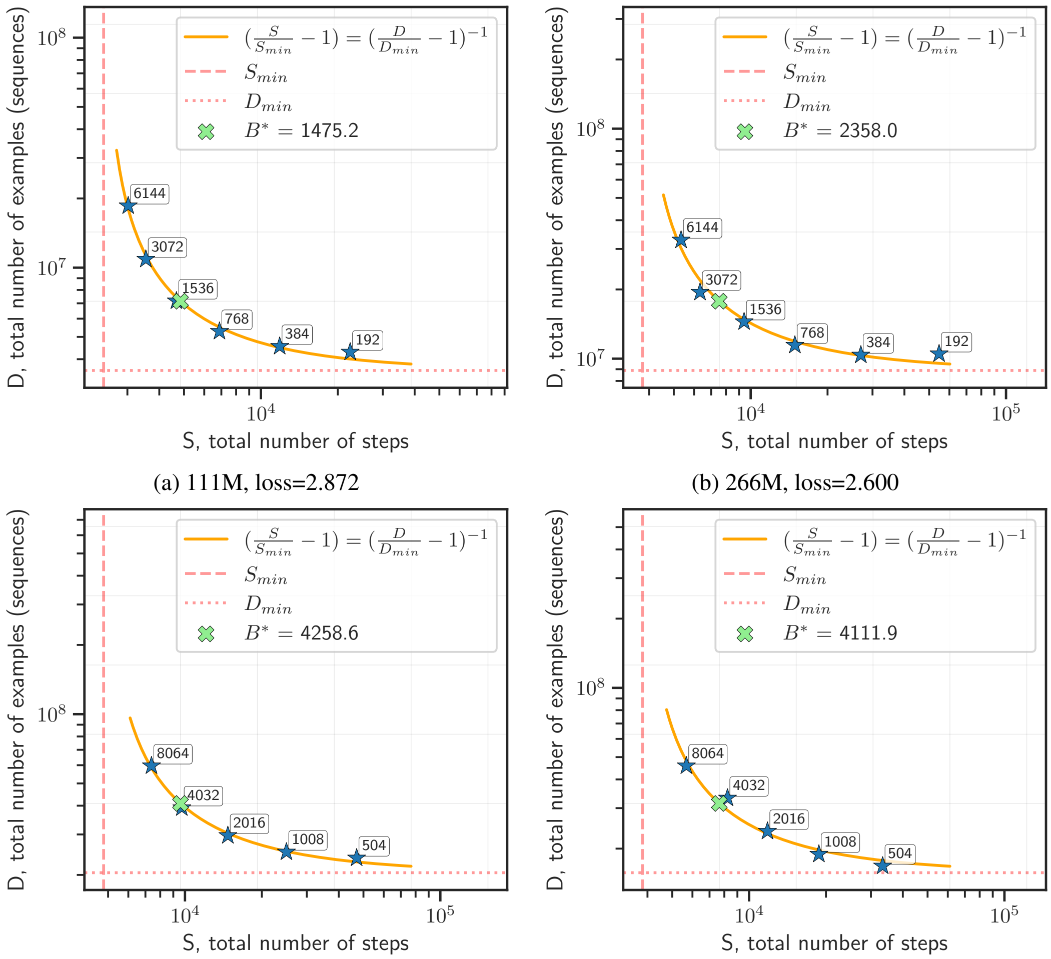
Fig. 9 illustrates all the fit scaling laws for the B_{crit} experiments. Notice that beyond the fitting points, the curves may not predict behavior well. In particular, we would expect all curves to eventually converge as D increases. Because loss targets beyond the fitting points are unreliable, we only compute B_{crit} at loss targets where all the data sizes can be estimated through interpolation.

Fig. 10 shows the specific fits of Eq. (5) at particular loss targets. While Eq. (5) reflects the data trend well over the given B values, we consistently find that points with very small B do not approach D_{min}. We discussed this observation further in Appendix B.

Finally, for clarity, we provide Algorithm 2, which gives the detailed approach to generating the B_{crit} power law in procedural form.

F.3 Estimating B_{crit} for the 3.3B model

To obtain an estimate of B_{crit} for the 3.3B model (shown in Fig. 1, *right*), it was not feasible to apply our full B_{crit} fitting procedure at this scale (i.e., fitting B-specific loss power-laws, etc.). Instead, we estimated B_{crit} based on two B, D_{min} pairs. That is, (based on an initial estimate of B_{crit}) we trained a 3.3B model to 23TPP with B=2016 and got a loss of 2.1688, and a separate model to 30TPP with B=4032, obtaining a loss of 2.1695. Given these losses are very close, these two models should have the same B_{crit} and thus same D_{min}.



(a) 111M, loss=2.872

(b) 266M, loss=2.600

(c) 610M scale, loss=2.437

(d) 1.7B scale, loss=2.295

Figure 10: **Example fits of trade-off Eq. (5) plotting S_{min} and D_{min}**: The empirical data has a good fit with Eq. (5) across model scales and loss targets. These estimates of B_{crit} are used in fitting the B_{crit}-scaling power law (Fig. 1 *right*).

We solve for this B_{crit} as follows. Let $B_2 = 4032$ and $B_1 = 2016$. By Eq. (6):

$$D_2 = D_{\text{min}}(1 + B_2/B_{\text{crit}}), \text{ and}$$
$$D_1 = D_{\text{min}}(1 + B_1/B_{\text{crit}})$$

Let $r = D_2/D_1$ (i.e., 30/23 in this case). If we divide the above equations, and solve for B_{crit}, we find:

$$B_{\text{crit}} = \frac{B_2 - rB_1}{r - 1}$$

Plugging in our values of r, B_1, and B_2, we obtain an estimated B_{crit} of 4610, corresponding to a D_{min} of approximately 16 TPP.

F.4 B_{crit} scaling in loss

Fig. 5 (*middle*) shows that B_{crit} is clearly not a power law in loss, as proposed in prior work [11, 4, 30]. However, if we only consider points with the same TPP, there does appear to be somewhat of a power-law relationship. In fact, this is implied by B_{crit} being a power law in D, along with the (standard) assumption that loss scales similarly in N and D.

Algorithm 2 Generating the B_crit power law

```
Initialize $bcrit\_scaling\_law\_fitting\_points = [\,]$
for $N$ in $model\_scales$ do
  ▷ Fit $B$-specific (and $N$-specific) scaling laws, $L_B(D)$:
  for $B$ in $batch\_sizes[N]$ do
    Reset $scaling\_law\_fitting\_points = [\,]$
    for $D$ in $10N, 20N, 80N, 320N, \ldots$ do
      Train $LLM(N, D, B)$, get validation loss $L'$
      $scaling\_law\_fitting\_points.add(\langle D, L' \rangle)$
    end for
    Fit $E_N$, $D_\text{const}$, $\beta$ for $L_B(D) = E_N + D_\text{const} D^{-\beta}$ on $scaling\_law\_fitting\_points$
  end for
  ▷ Use $L_B(D)$ to estimate $B_\text{crit}$ at various loss values:
  for $\hat{L}$ in $lossTargets[N]$ do
    Reset $tradeoff\_fitting\_points = [\,]$
    for $B$ in $batch\_sizes[N]$ do
      Get $D_B = L_B^{-1}(\hat{L}) = \left(\frac{D_\text{const}}{\hat{L} - E_N}\right)^{\frac{1}{\beta}}$
      $tradeoff\_fitting\_points.add(\langle D_B, S{=}D_B/B \rangle)$
    end for
    Fit $D_\text{min}$, $S_\text{min}$ for Eq. (5) on $tradeoff\_fitting\_points$
    $B_\text{crit} = D_\text{min}/S_\text{min}$
    $bcrit\_scaling\_law\_fitting\_points.add(\langle D_\text{min}, B_\text{crit} \rangle)$
  end for
end for
Fit $c$, $m$ for $B_\text{crit} = c(D_\text{min})^m$ on $bcrit\_scaling\_law\_fitting\_points$
```

Specifically, let $\hat{r} = D/N$ be the fixed TPP ratio. Therefore, $N = D/\hat{r}$. Assuming loss follows Eq. (8), we have:

$$\begin{aligned} L(N, D) &= E + N_\text{const} N^{-\alpha} + D_\text{const} D^{-\beta} \\ &= E + N_\text{const} \left(\frac{D}{\hat{r}}\right)^{-\alpha} + D_\text{const} D^{-\beta} \\ &= E + N_\text{const} \hat{r}^{\alpha} D^{-\alpha} + D_\text{const} D^{-\beta} \end{aligned}$$

Now, if $\alpha \approx \beta$, as is commonly accepted [3, 9, 80, 35, 81], we have:

$$\begin{aligned} L(D) &= E + (N_\text{const} \hat{r}^{\alpha} + D_\text{const}) D^{-\alpha} \\ &= E + K_\text{const} D^{-\alpha} \end{aligned}$$

where $K_\text{const} = N_\text{const} \hat{r}^{\alpha} + D_\text{const}$ is a constant. In other words, at a fixed TPP, loss is a power law in data. Given B_crit is also fundamentally a power law in data, then by the transitivity of power law relationships, B_crit is also a power law in loss in this context. This relationship can also be derived by expressing the D in Eq. (7) as a power law in B_crit, and substituting into $E + KD^{-\alpha}$.

F.5 Weight decay affects B scaling laws

Weight decay affects scaling of B_opt. Table 5 illustrates how fixing λ at different values alters the fitted B_opt power law. Larger λ systematically yields larger B_opt and poorer fit quality (lower R^2). When λ is fixed, the batch size that minimizes loss is only a *conditional optimum* (we denote it as $B_{\text{opt}|\lambda}$) because it compensates for suboptimal timescales τ rather than representing the globally tuned B_opt obtained when λ is optimized jointly.

The degraded R^2 values arise because a fixed λ forces B to balance two partially competing goals: (1) maintaining a good τ value, and (2) remaining below B_crit to avoid gradient redundancy. A simple power law cannot capture this coupled behavior.

From the relation $\tau = B/(\eta\lambda D)$ and the empirical scaling $\tau_\text{opt} \propto (D/N)^m$ (Eq. (3)), one can derive that, for constant η, λ, and N, the batch size preserving τ_opt should scale as $B \propto D^{m+1}$ (with $m + 1 \approx 0.47$ for our data). The exponents in Table 5, however, deviate from 0.47. For small D

Table 5: Different λ settings systematically affect fitted power laws for B_{opt}, and result in poorer fit quality (lower R^2). Fitted parameters change with λ and consequently projected B_{opt} values (in sequences) for different token budgets D.

Weight decay	Scaling law	R^2	D=1e10	D=1e11	D=1e12
0.4	$B_{\text{opt}} = 0.0006D^{0.607}$	0.706	587	2377	9615
0.2	$B_{\text{opt}} = 0.0012D^{0.543}$	0.926	323	1128	3937
0.1	$B_{\text{opt}} = 0.0123D^{0.429}$	0.972	240	644	1729
0.05	$B_{\text{opt}} = 0.384D^{0.270}$	0.689	192	358	667
0.025	$B_{\text{opt}} = 10.3D^{0.120}$	0.161	163	215	284
Tuned	$B_{\text{opt}} = 0.0306D^{0.383}$	*0.984*	*207*	*500*	*1207*

and large λ, the B that preserves τ_{opt} lies near or above B_{crit}, yielding worse loss due to gradient redundancy. This means $B_{\text{opt}|\lambda}$ is artificially lower for small D values. Since $B_{\text{opt}|\lambda}$ is affected differently for different D, the scaling law slope is distorted (in this case, increased). Analogous issues disrupt $B_{\text{opt}|\lambda}$ for small λ. These distortions reduce R^2 and impair generalization to large-scale training. When λ is tuned, this confound is removed, and the resulting B_{opt} law aligns cleanly with the expected $D^{0.4}$ scaling.

Weight decay affects scaling of B_{crit}. A similar confound arises for B_{crit}. Fixing λ causes deviations from τ_{opt} to mingle with true gradient-redundancy effects. At 111M scale and a target loss of 3.03, e.g., larger batches perform worse solely because τ drifts from its optimal value, reducing the fitted B_{crit} from 867 (tuned λ) to 707 (fixed λ=0.1). At other loss targets, B_{crit} is less affected. Since the estimated B_{crit} is affected differently at different scales, the slope of the B_{crit} scaling law is again artificially distorted. B_{crit} will appear to increase faster than $D^{0.5}$, and projections to larger scales will be inaccurate. In contrast, tuning λ isolates gradient-redundancy effects, yielding a stable $D^{0.5}$ relation that generalizes across scales.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Claims in the abstract and introduction are supported by extensive empirical results, culminating in the scaling law fits in Fig. 1.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are presented in Appendix B.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [NA]

Justification: While mathematical derivations are included in the appendices in order to support the empirical findings, there are no new theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: In Appendix C, we disclose the information needed to reproduce the experimental results in the paper. Model validation losses are all obtained through changes in configuration files (i.e., adjusting λ, η, B, N and D). Datasets and tokenizers are publicly-available and we disclose our standard model architecture settings and optimization hyperparameters in Appendix C.

   Guidelines:

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

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

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

Answer: [No]

Justification: Model validation losses are all obtained through changes in configuration files (i.e., adjusting λ, η, B, N and D) rather than code. Datasets and tokenizers are publicly-available and we disclose our standard model architecture settings and optimization hyperparameters in Appendix C.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: In Appendix C, we disclose the information needed to reproduce the experimental results in the paper. We disclose our standard model architecture and optimization hyperparameters in this section.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: We report statistical fit for our scaling laws in Sec. 2.4 and Sec. 3.3: R^2 values, and fitted parameter percentiles from a bootstrapping analysis (re-fitting the power law on 80% of points, 1000×, following Hoffmann et al. [3]). Random seed variance of validation losses are also reported in Appendix C.

   Guidelines:

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

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Appendix C reports the hardware used and training time details. Additionally, given the extensive experimental details and standard LLM settings, other practitioners can easily derive compute requirements based on their own specific hardware setting.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: This work conforms to the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Appendix A discusses the broader impacts of our work.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

Answer: [NA]

Justification: No models or datasets are released with our work.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: We cite and use all training and evaluation datasets in accordance with their licenses.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [NA]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

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