

---

# Small Batch Size Training for Language Models: When Vanilla SGD Works, and Why Gradient Accumulation Is Wasteful

---

**Martin Marek**  
New York University  
`martin.m@nyu.edu`

**Sanae Lotfi**  
New York University

**Aditya Somasundaram**  
Columbia University

**Andrew Gordon Wilson**  
New York University

**Micah Goldblum**  
Columbia University

## Abstract

Conventional wisdom dictates that small batch sizes make language model pretraining and fine-tuning unstable, motivating gradient accumulation, which trades off the number of optimizer steps for a proportional increase in batch size. While it is common to decrease the learning rate for smaller batch sizes, other hyperparameters are often held fixed. In this work, we revisit small batch sizes all the way down to batch size one, and we propose a rule for scaling Adam hyperparameters to small batch sizes. In particular, rather than holding the decay rate of the second moment fixed across batch sizes, we propose to hold its half-life fixed in terms of tokens. We find that small batch sizes (1) train stably, (2) are consistently more robust to hyperparameter choices, (3) achieve equal or better per-FLOP performance than larger batch sizes, and (4) notably enable stable language model training with vanilla SGD, even without momentum, despite storing no optimizer state. Building on these results, we provide practical recommendations for selecting a batch size and setting optimizer hyperparameters. We further recommend against gradient accumulation unless training on multiple devices with multiple model replicas. Finally, we show that a small batch size combined with an optimizer with a small state size can provide the performance benefits of full fine-tuning while maintaining a similar memory footprint to LoRA.

## 1 Introduction

Large batch sizes are widely believed to improve the stability of language model training [1, 2, 3, 4]. As a consequence, sophisticated optimizers that perform well in large-batch training are increasingly standard practice [5, 6, 7, 8, 9, 10]. In fact, it is common to simulate batch sizes even larger than the maximum batch size that fits into device memory through gradient accumulation [4, 3, 11, 12].

In small batch size pretraining experiments, one may observe loss spikes and heavy instability [13, 14, 15, 16]. While it is common to decrease the learning rate for smaller batch sizes, other hyperparameters, such as the decay rates for the first and second moments in Adam ( $\beta_1$  and  $\beta_2$ ), are often held fixed across batch sizes. We show that if instead of holding  $\beta_2$  fixed, we hold the *half-life* of  $\beta_2$  fixed (measured in number of tokens), stable training is possible all the way down to batch size one, as illustrated in [Figure 1a](#).

When scaling hyperparameters in this way, we find that small batch sizes can confer computational advantages and greater robustness. In the small batch regime, we observe that the speed of convergence is less sensitive to optimizer hyperparameters like the learning rate, and momentum becomes

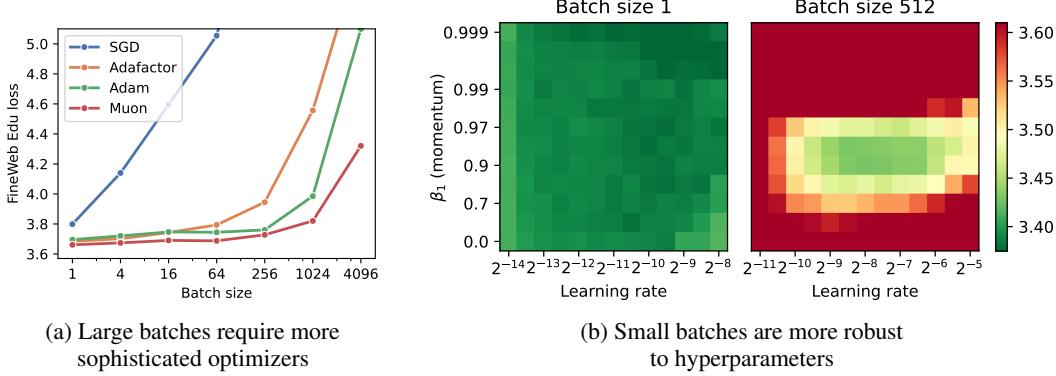


Figure 1: **Small batch sizes are more robust to optimizer design and hyperparameter values.** (a) Loss achieved by a transformer decoder-only language model with 30M active parameters trained on 600M tokens of FineWeb-Edu data using SGD, Adafactor, Adam, and Muon. At small batch sizes, all optimizers achieve similar loss; at large batch sizes, the gap between optimizers grows. (b) Loss achieved by GPT-2 (124M) trained on 2.5B tokens of FineWeb using AdamW when tuning the learning rate and  $\beta_1$  hyperparameters for batch sizes 1 and 512. While both batch sizes achieve a similar lowest loss, the smaller batch size is much more robust to the hyperparameter values, reducing the need for hyperparameter tuning. For batch size 512, we use the default hyperparameters from Brown et al. [17] and the nanoGPT public repository [18]. For batch size 1, we turn off weight decay and rescale  $\beta_2$  to preserve the token half-life, resulting in  $\beta_2 = 0.9999$ .

less necessary, as we find in Figure 1b. In contrast, large batch sizes require large learning rates to compensate for taking fewer steps at a given compute budget. Large step sizes in turn require that the optimizer make predictions about the loss surface far away from the current iterate. We find that predicting such far-away parameter updates requires a sophisticated and carefully tuned optimizer. For very small batch sizes, we find that even vanilla stochastic gradient descent (SGD), with no momentum or weight decay, becomes competitive for training language models, in contrast to findings from recent work [19]. The ability to perform stable training without momentum can confer significant memory advantages, since we do not need to store the corresponding optimizer state, which can have a significant memory footprint. Based on our observations, we also revisit Adafactor, a memory-efficient variant of Adam, and show that it can be a compelling alternative to Adam in the small batch regime, enabling training of larger models in a memory-constrained setting.

These findings could have a significant bearing on practice. Rather than perform gradient accumulation or use the largest batch size that fits into device memory, it could be preferable to use smaller batch sizes that enable simpler procedures, computational and memory advantages, and require less tuning. Our findings suggest a best practice of using the smallest batch size that maximizes model throughput. We find that these prescriptions hold in both LLM pretraining and fine-tuning.

It is common in convex optimization to use higher batch sizes as the loss converges to a minimum [1, 20]. Intuitively, the higher the gradient to gradient noise magnitude ratio, the smaller the batch size we should take. For example, if the gradient noise magnitude is close to zero, then we obtain roughly the same gradient estimate with a small batch size as a large batch size but pay far fewer floating-point operations (FLOPs), so small batch sizes are efficient. Smaller batch sizes allow us to take more steps for a fixed FLOP budget. As we converge to a minimum, that same ratio instead often tends to zero since the gradient norm vanishes while the gradient noise may not, in which case we may want a larger batch size. We hypothesize that language model training lives in the far-from-convergence regime where small batch sizes are efficient. Following compute-optimal scaling laws, we would not train language models into the convergence regime since we could achieve better performance by instead training a larger model on less data [21, 22].

The paper is structured as follows: in Sections 2 and 3, we cover the background and related work for language model training. In Section 4, we demonstrate that small batch sizes not only perform on par with larger batch sizes for several optimizers that we carefully tune, but they also exhibit more robustness to optimizer and hyperparameter choices. We also show that the gap between vanilla SGD and more sophisticated optimizers shrinks in the small batch regime. Moreover, we derive scaling heuristics for Adam’s hyperparameters and demonstrate that they yield improved performance when transferred to new, larger-scale settings. Finally, we apply these results to memory-efficient

fine-tuning using a small batch size and Adafactor, which achieves the best trade-off between memory footprint and performance. Motivated by our findings, we provide practical prescriptions for training language models in [Section 5](#). Namely, we recommend using the smallest batch size that maximizes model throughput and advise against gradient accumulation for most practitioners.

## 2 Background: Optimization for Language Models

In this section, we briefly review relevant optimization algorithms for language model training and the role of their hyperparameters.

**Stochastic gradient descent (SGD)** [23, 24]. SGD updates model parameters by computing gradients on mini-batches of data as follows:  $\theta_{t+1} = \theta_t - \eta g_t$ , where  $\theta_t$  are the model parameters at step  $t$ ,  $\eta$  is the learning rate, and  $g_t = \nabla_{\theta} \mathcal{L}(\theta_t)$  is the gradient of the loss function with respect to  $\theta_t$  averaged over  $B$  samples. We refer to  $B$  as the batch size and consider the extreme case of  $B = 1$  in our experiments.

We show that although SGD is extremely simple, it can perform competitively when correctly tuned at small batch sizes. Momentum is often added to accumulate a moving average of past gradients, smoothing updates, and accelerating convergence. However, we focus on vanilla SGD in our experiments to make the point that, with a small batch size, even momentum is unnecessary.

**Adam** [25]. Adam is an adaptive optimizer that maintains an exponential moving average (EMA) of the gradient and squared gradient, referred to as the first and second moments, and denoted as  $m_t$  and  $v_t$ . The timescales of these moving averages are controlled by decay rates  $\beta_1$  and  $\beta_2$ :

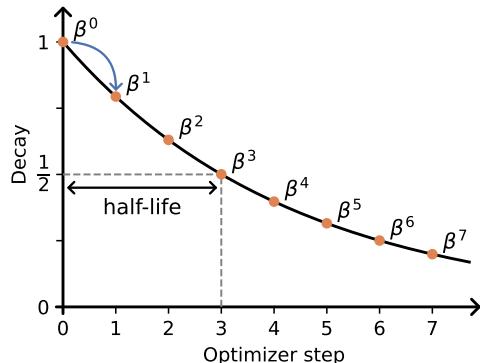
$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \end{aligned} \tag{1}$$

The model parameters are then updated as follows:  $\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t + \epsilon}}$ , where  $\epsilon$  is a small constant to prevent division by zero.

Higher values of  $\beta_1$  and  $\beta_2$  place more weight on past gradients, meaning that the moments are averaged over longer timescales and evolve slowly. When training language models, values like  $\beta_1 = 0.9$  and  $\beta_2 \in [0.95, 0.98]$  are often used in practice [17, 26]. While most researchers and practitioners do not scale  $\beta_1$  and  $\beta_2$  with the batch size, we show in our work that scaling  $\beta_2$  in particular is crucial for achieving good performance with Adam at a small batch size.

**Moment half-life.** We found it helpful across many of our experiments to measure the typical timescales (measured in number of tokens) that the first and second moments in Adam are averaged over, instead of directly working with  $\beta_1$  and  $\beta_2$ .

At each Adam update step [eq. (1)], the contribution of previous gradients to the first moment gets reduced by a factor of  $\beta_1$  and the contribution to the second moment gets reduced by a factor of  $\beta_2$ . Hence, by definition, there exists a certain number of steps  $n$ , after which the contribution of any mini-batch gradient is halved:  $\beta^n = \frac{1}{2}$ . Since every update step corresponds to observing  $(B \cdot T)$  tokens, where  $B$  is the batch size and  $T$  is the sequence length, we can convert the number of optimizer steps into a number of observed tokens. As such, we can express the number of tokens it takes to decay a mini-batch gradient by a factor of  $\frac{1}{2}$  as the *half-life* of the decay rate, denoted by  $t_{1/2}$ , where  $\beta^{\frac{t_{1/2}}{BT}} = \frac{1}{2}$ . The half-life provides a measure of the “typical” timescale that gradients are averaged over. For



**Figure 2: Moment half-life.** The first and second moments in the Adam optimizer are exponential moving averages of past mini-batch gradients. At each step, the contribution of past mini-batch gradients decays by a factor of  $\beta$ . After a given number of optimizer steps (or equivalently, after a given number of training tokens), the contribution of any mini-batch gradient will decay by a factor of  $\frac{1}{2}$ . We call this number of tokens the decay *half-life*.

example, if we use a half-life of  $t_{1/2} = 10M$  tokens, this means that the second moment is averaged over *roughly* 10M tokens. Throughout this paper, we will refer to the half-life of the first moment as  $t_1$  and the half-life of the second moment as  $t_2$ , analogously to the more conventional parameterization  $\beta_1$  and  $\beta_2$ .

**Adafactor** [27]. The second moment estimate  $v_t$  used in Adam requires storing a number of floating point values equal to the parameter count of the model itself. Instead, Adafactor only stores the per-row and per-column sums of the moving average for the neural network weight matrices and estimates the per-parameter second moment using these sums. Adafactor also does not store a first moment estimate at all. Therefore, for a weight matrix with shape  $d_1 \times d_2$ , the memory requirements for the moving averages are reduced from  $2 \times d_1 \times d_2$  for Adam to  $d_1 + d_2$  for Adafactor. We demonstrate in Section 4.5 that thanks to this sublinear memory cost, Adafactor achieves the best performance–memory trade-off.

**Muon** [9]. Another optimizer that was recently demonstrated to be competitive for language model training is MomentUm Orthogonalized by Newton-Schulz (Muon). Muon is specifically designed for  $\geq 2$ -dimensional parameter tensors, such as weight matrices in linear layers, where the Newton-Schulz iterative method is applied to the first moment estimate  $m_t$  before using it to update the parameters. Muon is often used in conjunction with other optimizers like Adam. In our experiments using Muon, we only apply it to hidden layers and use Adam for input embedding and final layers.

**Gradient accumulation.** When hardware memory constraints prevent us from using large batch sizes directly, we can simulate a large batch size by summing gradients across multiple successive micro-batches and only periodically performing an optimizer step. Under gradient accumulation, the effective batch size is equal to the micro-batch size multiplied by the number of gradient accumulation steps. While gradient accumulation allows for larger effective batch sizes, it still increases memory usage compared to using a small batch size, because it requires storing the accumulated gradients.

### 3 Related Work

Batch size has long been studied in both theory and practice: its impact on the optimization path [28, 29, 30, 31], its interaction with different optimizers [32, 33, 34, 35], and its effect on the optimal learning rates and momentum hyperparameters [36, 37, 38, 39, 34, 40, 41, 42, 41, 43, 44]. Prior work has also explored related themes, such as the flatness argument favoring SGD [45, 46, 47, 48], SGD’s implicit biases [49, 50, 51, 52], and the comparison between stochastic and full-batch optimization [53, 54, 55, 56]. However, modern language model training is neither convex nor characterized by multiple epoch training like vision settings. In fact, we often train for a limited token budget and stop far before convergence, since otherwise scaling laws suggest that we could train a much better language model for the same compute budget by training a larger model on fewer tokens [22]. Hence, prior intuitions, such as the need for a larger batch size in the convergence phase, may not apply. For this reason, in the rest of this section, we cover works that focus on modern language model training specifically.

Zhang et al. [41] and Shallue et al. [34] examine the critical batch size, which represents the threshold beyond which greater data parallelism would lead to diminishing returns, and tune all optimizer hyperparameters. They argue that any batch size below the critical batch size should perform equally well when training for the same number of tokens. Differently from these works, we empirically demonstrate that small batch sizes, including the extreme value of batch size one, perform on par with or even better than larger batch sizes, and we also show that small batches are more robust to optimizer and hyperparameter choices and enable SGD to perform competitively. We validate these findings on language models with modern design choices and up to 1.3 billion parameters.

On the other hand, Vyas et al. [57] argue that gradient noise from small batch sizes does not yield similar performance advantages in the single epoch setting as it does in multiple-epoch training. Experimental results from Filatov et al. [58] also suggest that small batches do not work as well as large batches for language models trained on C4 [59], but notably, the authors do not tune the decay rates  $\beta_1$  and  $\beta_2$  of the Adam optimizer. In the same vein, Xiao [14] claim that both excessively small and large batch sizes underperform for language model training with Adam, leading to a U-shaped curve of the loss as a function of the batch size. We reproduce the exact experiments performed by Xiao [14] and show that their conclusions are an artifact of incomplete hyperparameter tuning; we

find that simply using a slower decay rate of the second moment estimate  $\beta_2$  makes small batch sizes competitive.

Zhao et al. [19] train language models on C4 using different diagonal preconditioning optimizers, namely Adam, Lion, Adafactor with momentum, SGD with momentum, and signed SGD with momentum. They demonstrate that all these optimizers, aside from SGD which lags behind, achieve comparable optimal performance and robustness to hyperparameters. We challenge this conclusion in our work and argue that in the small batch size regime, SGD can perform on par with Adam.

Shallue et al. [34], Kidambi et al. [40] and Zhang et al. [41] find that momentum is not necessary for SGD when the batch size is small, which aligns with our findings. However, Kidambi et al. [40] focus on small-scale vision experiments, Shallue et al. [34] and Kidambi et al. [40] do not compare the performance of SGD vs. Adam, and Zhang et al. [41] claim that SGD performs significantly worse than Adam. We hypothesize this observation could be a result of Zhang et al. [41] using 64 as the smallest batch size – we test batch sizes all the way down to 1. Zhang et al. [35] highlight that the benefits of more sophisticated optimizers diminish as batch size decreases, but their theoretical results assume a convex quadratic objective and their empirical results only cover a single language modeling experiment that is limited to a two-layer transformer.

Porian et al. [42] and Zhang et al. [41] find that increasing the value of  $\beta_2$  is important for small-batch training with Adam. However, their approach is based on discrete hyperparameter sweeps across  $\beta_2$ . In contrast, we introduce and validate a principled scaling rule that holds the second-moment half-life constant in terms of tokens. Busbridge et al. [60] propose a scaling rule for the decay coefficient of exponential moving averages similar to ours, but applied in the context of model weight averaging. Zhang et al. [44] show that there exists a threshold of  $\beta_2^*$  such that when  $\beta_2 > \beta_2^*$ , Adam converges (this threshold decreases with batch size, i.e. the threshold is higher for smaller batch sizes). Chowdhery et al. [43] note that increasing  $\beta_2$  during training leads to better performance for rare token embeddings. While this is an important insight, we provide a simpler batch-size-aware heuristic that applies even when using a constant  $\beta_2$  throughout training.

## 4 Experimental Results

Recent works find that larger batch sizes and relatively sophisticated optimizers are necessary for stable language model training. For instance, Zhao et al. [19], Kunstner et al. [61], and Zhang et al. [62] observe that SGD has an extremely slow convergence speed for training language models in comparison to adaptive optimizers like Adam. Other works find that Adam performs poorly with small batch sizes [57, 14, 63, 64].

In this section, we revisit and challenge these beliefs by noting that such findings arise due to poor hyperparameter choices and specific experimental setups, for example, not adjusting the decay parameters  $\beta_1$  and  $\beta_2$  for Adam at small batch sizes, or using too large a batch size for SGD. We put these beliefs to the test by running a thorough grid search over optimizer hyperparameters and batch sizes for SGD, Adam, Adafactor, and Muon. As the cost of a grid search grows exponentially with the number of hyperparameters, we only perform exhaustive grid searches on a small transformer decoder-only model with 30 million active parameters following Liu et al. [65] and Xiao [14]. We use these results to propose heuristics for scaling hyperparameters across batch sizes, and we validate our findings at a larger scale by pretraining GPT-2 (124M) [66] and GPT-3 (1.3B) [17] and fine-tuning Gemma 3 (4B) [67]. In our experiments, we study batch sizes spanning more than four orders of magnitude:  $\{1, 4, 16, 64, 256, 1024, \text{ and } 4096\}$ .

### 4.1 Small Batch Sizes Render Sophisticated Optimizers Unnecessary

We train the 30M model on 600 million tokens from the FineWeb-Edu dataset [68], following Chinchilla scaling laws [22]. Our model adopts modern design choice, including rotary embeddings [69], QK-normalization [70], RMSNorm [71], GELU activations [72], and separate input and output embeddings. We use a context length of 512 and tokenize the dataset using the GPT-2 tokenizer. We provide additional experimental details in [Appendix A](#).

To verify the effect of the batch size and optimizer choice on the performance of our model, we run a dense grid search over all the hyperparameters of SGD, Adam, Adafactor, and Muon. In particular,

we tune the learning rate and the decay factors ( $\beta_1, \beta_2$ ) jointly for each batch size to obtain the hyperparameter configuration that provides the lowest validation loss.

The results in [Figure 1a](#) highlight that all optimizers perform best at the smallest batch size, and as the batch size increases, not only does the performance of each optimizer degrade but also the gap between different optimizers widens.

*Why do large batches require more sophisticated optimizers?* Every time our optimizer takes a step, it makes a prediction about the loss function away from the current parameter vector. When we train with a higher learning rate and take larger steps, for example in large batch training, our optimizer must make predictions about the loss function farther away from the current parameter vector. We hypothesize that taking larger steps leads to a harder prediction problem and that this harder prediction problem requires a more sophisticated or better-tuned optimizer.

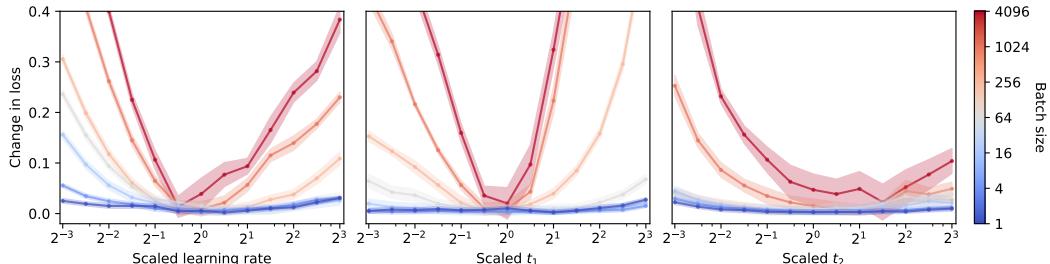
*Why is momentum less necessary for small batch sizes?* Momentum can be seen as a means of dampening oscillations that occur on ill-conditioned loss functions [73]. When we take large steps, we may overshoot the minimum across short (i.e. high curvature) axes, leading to oscillations. By averaging gradients across iterations, these oscillations cancel out in the momentum term. However, when the optimizer takes small steps, as is the case in small batch training paired with a small learning rate, parameter updates do not overshoot the minimum across short axes, so there are no oscillations to damp. We illustrate this effect in [Appendix B](#) through a toy example.

**Summary:** The difference in performance between optimizers shrinks under small batch sizes. In fact, vanilla SGD without momentum performs competitively for small batch sizes.

## 4.2 Large Batch Training is Sensitive to Hyperparameters

To evaluate robustness to hyperparameters at each batch size, we perform an ablation around the optimal hyperparameters achieving the lowest validation loss for the Adam optimizer. We use the same 30M parameter model trained on the FineWeb-Edu dataset as in [Section 4.1](#).

We report in [Figure 3](#) the change in the loss with respect to the lowest value of the loss achieved at every batch size, ranging from 1 to 4096. On the x-axis, we report the scaled value of the hyperparameters relative to the value of the hyperparameter that achieved the best loss. We perform this rescaling to make the plots for different batch sizes easier to compare, since the scale of their optimal hyperparameters varies.



**Figure 3: Small batch sizes are robust to hyperparameter misspecification.** Loss sensitivity to learning rate,  $\beta_1$ , and  $\beta_2$  using Adam at batch sizes from 1 to 4096. Each curve sweeps a single hyperparameter while holding the others fixed at their optimal values. We observe that smaller batch sizes exhibit broader low-loss regions across all hyperparameters, indicating greater robustness to misspecification. The x-axis shows a scaling of each hyperparameter relative to its optimum value. We parameterize  $\beta_1$  and  $\beta_2$  in terms of their decay half-lives, as described in [Section 2](#). Results are shown for the 30M parameter model trained on 600M tokens of FineWeb-Edu.

We observe a striking result: small batch sizes are significantly more robust to *all* of Adam’s hyperparameters than large batch sizes. More importantly, batch size one achieves a nearly optimal loss for the entire range of learning rates,  $\beta_1$ , and  $\beta_2$  hyperparameters that we consider in our search, whereas the loss for larger batches increases sharply as we vary the hyperparameter values.

To highlight the compounding value of this robustness displayed by small batch sizes as the number of hyperparameters increases, we perform a joint 2-dimensional ablation with respect to both the learning rate and decay rate  $\beta_1$  on a GPT-2 model in [Figure 1b](#). Consistent with the previous result, batch size one yields much broader low-loss regions across both hyperparameters compared to batch size 512. Therefore, if practitioners were to consider not only the budget to run a single training run but also the computational budget for tuning the corresponding optimizer hyperparameters, a smaller batch size might be preferable.

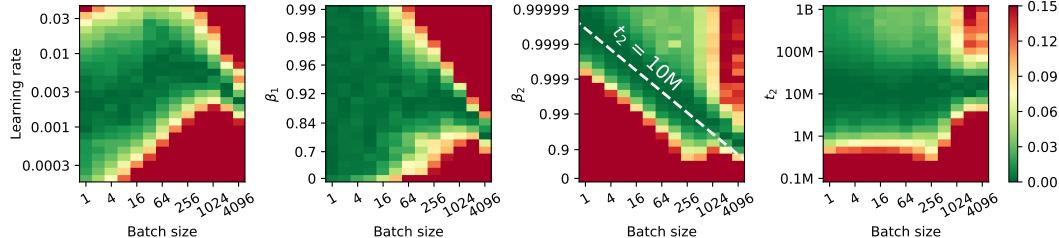
*Why are smaller batch sizes more robust to hyperparameter choices?* Similar to the previous subsection, we hypothesize that large batch sizes require careful hyperparameter tuning since they require large step sizes and therefore have to make predictions about the loss surface further away from the current parameter vector.

**Summary:** Small batch sizes are more robust to hyperparameter misspecification and might be preferable after accounting for the hyperparameter tuning budget.

### 4.3 Adam Hyperparameters: How to Scale Them with Batch Size and Why That Is Necessary

We use our grid search results on the 30M model to test existing scaling laws for Adam hyperparameters and also devise new scaling heuristics for these hyperparameters. [Figure 4](#) shows the validation loss on FineWeb-Edu for different Adam hyperparameters, namely the learning rate, and decay parameters  $\beta_1$  and  $\beta_2$ . We similarly plot the loss for different values of the second moment half-life  $t_2$ , which refers to the number of tokens it takes for a gradient’s contribution to the momentum to be reduced to half of its initial value. We provide a more detailed description of  $t_2$  in [Section 2](#).

For the learning rate, we observe that the square root scaling recommended by several works and commonly used in practice does not hold [74, 7, 37, 75, 76]. In fact, [Figure 4](#) (left) shows that the learning rate scales more slowly than a square root factor with the batch size. For instance, as we scale the batch size from 1 to 1024, the square root rule would indicate that the corresponding learning rate should be scaled by a factor of 32, whereas we find that a factor of only about 3 empirically works better.



**Figure 4: Fixing the half-life of the second moment estimate in terms of tokens  $t_2$  scales better than fixing  $\beta_2$ .** We plot the validation loss on FineWeb-Edu for the 30M parameter model when varying the Adam learning rate,  $\beta_1$ ,  $\beta_2$ , and the second moment half-life  $t_2$ , across different batch sizes. The learning rate does not follow the commonly recommended square root scaling with batch size. We also observe that the default  $\beta_1 = 0.9$  performs well across batch sizes. In contrast, keeping  $\beta_2$  fixed leads to suboptimal performance at small batch sizes. Instead, fixing the second moment half-life  $t_2$  across batch sizes provides a simple and effective scaling rule.

We also validate that the default value of the first moment decay  $\beta_1 = 0.9$  used by practitioners for language model training does indeed achieve good performance across batch sizes. In contrast, keeping  $\beta_2$  fixed to its default value, typically in the range  $[0.95, 0.98]$  for language models, does not scale effectively to smaller batches, which require a significantly larger value of  $\beta_2$ . On the other hand, measuring the decay of the second moment in tokens through the half-life  $t_2$  offers a strong scaling heuristic: keeping  $t_2$  fixed to its optimal value as we scale up or down the batch size provides good performance without re-tuning this hyperparameter at new batch sizes. This finding translates to the following scaling heuristic: if we were to scale the batch size from  $B$  to  $B^*$ , the new  $\beta_2^*$  would depend on the old  $\beta$  associated with  $B$  as follows:

$$\beta_2^* = \beta_2^{(B^*/B)} \quad (2)$$

Next, we test our scaling heuristics in a different setting proposed in Xiao [14]. The authors train a decoder-only transformer with 19M non-embedding parameters using Adam on the C4 dataset [59], and only tune the learning rate with  $\beta_1$  and  $\beta_2$  fixed to their default values of 0.9 and 0.95, respectively. We reproduce their Figure 10(a) results in Figure 5 (left), where small batch sizes of 16 and 32 perform worse than larger batch sizes of 128 and 256.

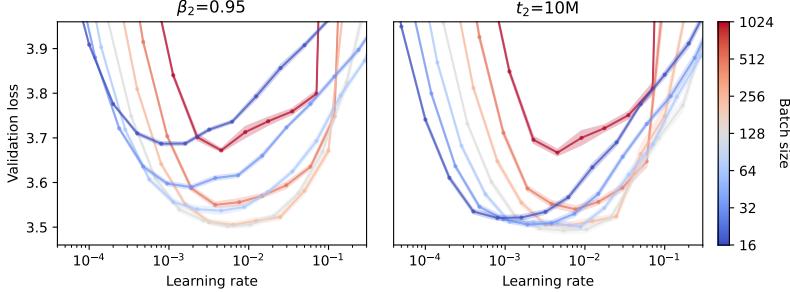


Figure 5: **Scaling  $\beta_2$  according to our heuristics significantly improves the performance of small batch sizes.** **Left:** Results from Xiao [14], showing suboptimal performance for small batch sizes due to fixed Adam hyperparameters. **Right:** Our results after scaling  $\beta_2$  such that the decay half-life  $t_2$  is constant across batch sizes. The smallest batch sizes now perform on par with larger batch sizes.

While our scaling heuristics imply that  $\beta_1 = 0.9$  could work well across batch sizes, we recommend scaling  $\beta_2$  such that  $t_2$  is fixed. By following this simple one-shot scaling recommendation and without any additional hyperparameter tuning, we achieve significantly different results in Figure 5 (right). Strikingly, small batch sizes 16 and 32 achieve on par performance with larger batch sizes 128 and 256, therefore implying that: (i) the previous results from Xiao [14] are largely a consequence of incomplete hyperparameter tuning, (ii) our simple one-shot heuristic generalizes to other settings.

**Summary:** While keeping  $\beta_1$  fixed to the default value of 0.9 scales well with the batch size,  $\beta_2$  should be scaled such that the token half-life is fixed. With this prescription, small batch sizes can perform on par with or better than larger batch sizes.

#### 4.4 SGD Performs Competitively with Adaptive Optimizers at the Billion+ Parameter Scale

Through our experiments with 30M parameter models, we conclude that: (1) small batch sizes can perform on par with or better than larger batch sizes, (2) small batch sizes require less sophisticated optimizers and are more robust to hyperparameter misspecification, (3) in particular, vanilla SGD can be competitive with adaptive optimizers in the small batch regime, and (4) we propose a new scaling heuristic for  $\beta_2$  such that the half-life  $t_2$  is fixed, and validate that it leads to improved performance. In Figure 6, we test these findings at a larger scale by training GPT-2 (124M) [66] and GPT-3 (1.3B) [17] on the FineWeb dataset [68].

For the larger 1.3B model, hyperparameter tuning of the learning rate for each optimizer was computationally prohibitive; therefore, we used AdamW with the default hyperparameter settings from Brown et al. [17] (batch size 512,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.95$ ,  $w = 0.1$ ) as our baseline. When using batch size 1 with Adam, we do not perform any hyperparameter tuning – we turn off weight decay and scale the learning rate following our empirical results from Section 4.3. We compare two scaling heuristics for  $\beta_2$ : keeping  $\beta_2$  fixed and keeping the half-life  $t_2$  fixed. We do, however, sweep over four learning rates for Adafactor and SGD, since we have no reference values for these optimizers. Interestingly, we find that SGD performs on par with the AdamW baseline, and Adam and Adafactor with batch size 1 both outperform the AdamW baseline.

For the smaller (124M) model, we follow the same procedure, except we tune the learning rate of each optimizer, which improves the performance of the AdamW baseline.

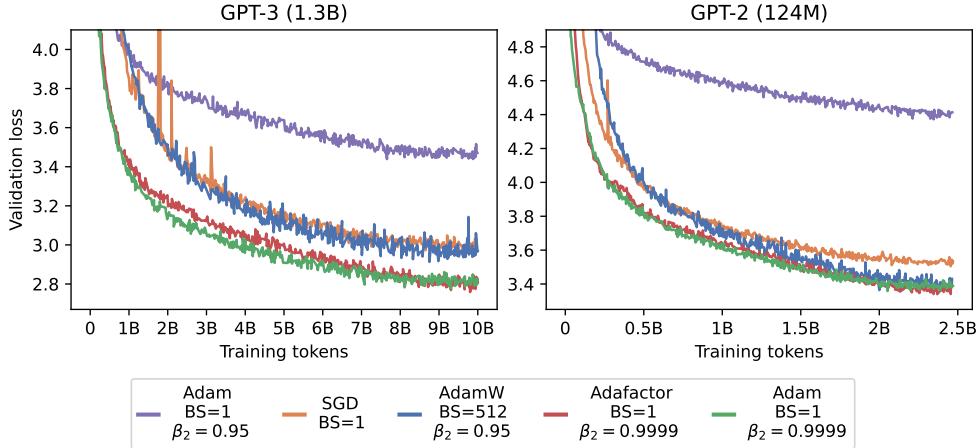


Figure 6: **Vanilla SGD performs competitively at larger model scales with minimal tuning.** We train GPT-3 (1.3B) and GPT-2 (124M) on FineWeb and compare validation loss across optimizers. **Left:** For GPT-3 (1.3B), vanilla SGD with batch size 1 and no momentum performs on par with the AdamW configuration from Brown et al. [17], despite having no optimizer state. Our proposed  $\beta_2$  scaling rule improves Adam’s performance at small batch sizes and outperforms the baseline. **Right:** After tuning the learning rate of each optimizer on GPT-2 (124M), Adam and Adafactor at batch size 1 match the performance of AdamW at batch size 512, while SGD performs slightly worse.

#### 4.5 Full Fine-tuning on a Budget

In Sections 4.1 to 4.4, we studied pretraining across different model scales and discovered that with small batch sizes, optimizers with small or even no state (like Adafactor and SGD) can perform competitively with Adam. We now exploit this observation for memory-efficient fine-tuning.

While pretraining is typically bottlenecked by compute, fine-tuning is instead often bottlenecked by memory. Indeed, fine-tuning a large model on a small dataset does not require many FLOPs, but it does require storing the (large) model and optimizer state in memory. In Figure 7, we show an example where the GPU has enough memory to store the model, but not enough memory to also store an expensive optimizer like Adam.

A common approach to reduce memory usage during fine-tuning is to freeze the model weights and only train Low-Rank Adaptation (LoRA) [77] modules on top of the frozen model, which substantially reduces the number of trainable parameters and also proportionately reduces the size of the optimizer state. However, LoRA often underperforms full parameter fine-tuning [78, 79].

We propose to use a small batch size combined with a small optimizer like Adafactor to get the performance benefits of full fine-tuning while having similar memory requirements to LoRA.

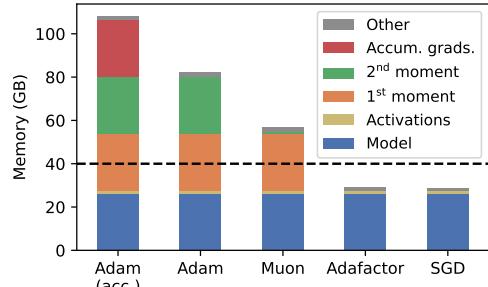


Figure 7: **Minimum memory requirements** to train GPT-3 (13B), assuming all parameters and activations are stored in 16-bit precision, gradient checkpointing is used after every transformer block layer, and the backward pass is fused with the optimizer update step (so that the full model gradient never needs to be materialized in memory). The dashed line shows the available memory on an NVIDIA A100 40GB GPU, which is not large enough to store expensive optimizer states or accumulated gradients. We measured the true memory usage of each optimizer by performing fused backward passes in PyTorch on an NVIDIA B200 GPU.

We provide an example in Figure 8 by fine-tuning a non-instruction-tuned Gemma 3 (4B) on the MATH dataset for 5 epochs. As a baseline, we fine-tune the model with Adam and batch size 16 (requiring gradient accumulation), storing all parameters in float32 precision, which requires storing a total of 16 bytes in memory per model parameter. Another baseline is provided by LoRA, which requires only around 2 bytes per model parameter since the pretrained model checkpoint is stored in bfloat16 precision, and the number of trainable parameters for LoRA is negligible compared to the model size.

We make two observations based on our results. First, when using Adam, we can reduce the batch size all the way down to one while following the  $\beta_2$  scaling heuristic from Equation (2), eliminating the need to store accumulated gradients in memory. Second, when a small batch size is used, we can replace Adam with Adafactor, which has a much smaller state size. To further reduce the memory use of Adafactor, we store its (tiny) optimizer state in float32, but we store model weights in bfloat16, using stochastic rounding to update the weights after every optimizer step (we discuss the importance of stochastic rounding in Appendix A.3).

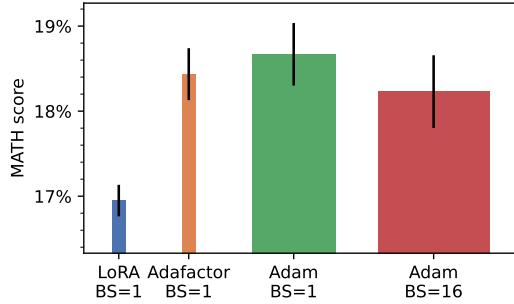


Figure 8: **Adafactor enables memory-efficient fine-tuning.** We fine-tune and evaluate a non-instruction-tuned Gemma 3 (4B) model [67] on the MATH dataset [80]. We compare the use of LoRA ( $r = 16$ ) against full model fine-tuning, and a small vs. large batch size. For each training method, the width of the plotted bar is proportional to the memory footprint. LoRA uses bfloat16 model weights and float32 adaptors, and Adafactor similarly uses bfloat16 weights but float32 optimizer state. We fix the second moment half-life to be consistent across batch sizes ( $\beta_2 = 0.95$  for batch size 16 and  $\beta_2 = 0.997$  for batch size 1). We run each optimizer for 5 epochs and sweep over 8 learning rates. We train LoRA using Adam.

**Summary:** We recommend that practitioners training large models in memory-constrained settings exploit the benefits of small batch sizes (e.g. using optimizers with a small state size) rather than trying to emulate the large batch size setting typically used in industry.

## 5 Practical Recommendations and Discussion

We show that small batch sizes, even as low as 1, consistently match or outperform large batches across model scales when tuned properly, in both pretraining and fine-tuning. In particular, SGD with batch size 1 performs on par with the AdamW baseline for GPT-3 (1.3B) with minimal tuning. We also propose a simple scaling rule for  $\beta_2$  based on a fixed second-moment half-life in tokens, which generalizes across model scales. Our results contradict common assumptions about large batch training and gradient accumulation.

Our general recommendation for setting the batch size is to **use the smallest batch size that maximizes model throughput** (measured in tokens / second) or equivalently the model FLOPs utilization (MFU) [43]. To achieve this objective, the accelerator must spend more time performing computation than moving data between HBM and register memory, which in practice means using a batch size of at least several hundred tokens per device [81]. When training frontier models on large clusters, this can correspond to a batch size of many millions of tokens. But when training on a single device, the smallest batch size required to achieve high arithmetic intensity could be as low as a few thousand tokens. We therefore recommend **avoiding gradient accumulation** unless training on multiple devices with multiple model replicas, where the bandwidth between model replicas is a bottleneck. Lastly, when working in a memory-constrained setting, we recommend **exploiting the benefits of simple optimizers combined with a small batch size**.

Several open questions remain. How do these findings interact with batch size schedules? Can we further reduce weight precision in fine-tuning from 16 to 8 or even 4 bits? Can we design more effective optimizers with a small state, tuned to the small batch size regime? And can we theoretically understand why fixing the second-moment half-life in tokens generalizes across scales?

## Acknowledgements

We thank Shikai Qiu and Alexandra Souly for helpful discussions. This work was supported by NSF CAREER IIS-2145492, NSF CDS&E-MSS 2134216, NSF HDR-2118310, BigHat Biosciences, and Google’s TPU Research Cloud (TRC) program: <https://sites.research.google/trc/>.

## References

- [1] Nidham Gazagnadou, Robert Gower, and Joseph Salmon. Optimal mini-batch and step sizes for saga. In *International conference on machine learning*, pages 2142–2150. PMLR, 2019.
- [2] Atsushi Nitanda, Tomoya Murata, and Taiji Suzuki. Sharp characterization of optimal minibatch size for stochastic finite sum convex optimization. *Knowledge and Information Systems*, 63(9):2513–2539, 2021.
- [3] Sungkyung Kim, Adam Lee, Junyoung Park, Sounho Chung, Jusang Oh, and Jay-Yoon Lee. Parameter-efficient fine-tuning of instructblip for visual reasoning tasks. In *Efficient Natural Language and Speech Processing Workshop at NeurIPS 2023*, 2023. URL [https://neurips2023-enlsp.github.io/papers/paper\\_88.pdf](https://neurips2023-enlsp.github.io/papers/paper_88.pdf).
- [4] Deepanway Ghosal, Yew Ken Chia, Navonil Majumder, and Soujanya Poria. Flacuna: Unleashing the problem solving power of vicuna using flan fine-tuning. *arXiv preprint arXiv:2307.02053*, 2023.
- [5] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.
- [6] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018.
- [7] 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.
- [8] 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.
- [9] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng Yo, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL <https://kellerjordan.github.io/posts/muon/>.
- [10] 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.
- [11] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [12] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.
- [13] Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J.D. Prince, and Yanshuai Cao. Optimizing deeper transformers on small datasets. *arXiv preprint arXiv:2012.15355*, 2020.
- [14] Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling. *arXiv preprint arXiv:2409.15156*, 2024.

[15] Maxime Labonne. Fine-tune llama 3.1 ultra-efficiently with unsloth. <https://huggingface.co/blog/mlabonne/sft-llama3>, 2024. Accessed: 2025-05-15.

[16] Weights & Biases. Comprehensive guide to llm pre-training steps. [https://wandb.ai/site/articles/training\\_llms/pre-training-steps/](https://wandb.ai/site/articles/training_llms/pre-training-steps/), 2025. Accessed: 2025-05-15.

[17] 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.

[18] Andrej Karpathy. nanoGPT: The simplest, fastest repository for training/fine-tuning medium-sized gpts., 2023. URL <https://github.com/karpathy/nanoGPT>.

[19] Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham M Kakade. Deconstructing what makes a good optimizer for autoregressive language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[20] Scott Sievert and Shrey Shah. Improving the convergence of sgd through adaptive batch sizes. *arXiv preprint arXiv:1910.08222*, 2019.

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

[22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *Advances in neural information processing systems*, 2022.

[23] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[24] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[25] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[27] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[28] Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of dnn loss and the sgd step length. *arXiv preprint arXiv:1807.05031*, 2018.

[29] Stanisław Jastrzębski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. The break-even point on optimization trajectories of deep neural networks. *arXiv preprint arXiv:2002.09572*, 2020.

[30] Kiwon Lee, Andrew Cheng, Elliot Paquette, and Courtney Paquette. Trajectory of mini-batch momentum: Batch size saturation and convergence in high dimensions. *Advances in Neural Information Processing Systems*, 35:36944–36957, 2022.

[31] Alexander Atanasov, Alexandru Meterez, James B Simon, and Cengiz Pehlevan. The optimization landscape of sgd across the feature learning strength. *arXiv preprint arXiv:2410.04642*, 2024.

[32] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural networks*, 16(10):1429–1451, 2003.

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

[34] 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.

[35] 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.

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

[37] Diego Granziol, Stefan Zohren, and Stephen Roberts. Learning rates as a function of batch size: A random matrix theory approach to neural network training. *Journal of Machine Learning Research*, 23(173):1–65, 2022.

[38] Runzhe Wang, Sadhika Malladi, Tianhao Wang, Kaifeng Lyu, and Zhiyuan Li. The marginal value of momentum for small learning rate sgd. *arXiv preprint arXiv:2307.15196*, 2023.

[39] Jingwen Fu, Bohan Wang, Huishuai Zhang, Zhizheng Zhang, Wei Chen, and Nanning Zheng. When and why momentum accelerates sgd: An empirical study. *arXiv preprint arXiv:2306.09000*, 2023.

[40] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2018. doi: 10.1109/ITA.2018.8503173.

[41] 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.

[42] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=4fSSqpk1sM>.

[43] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[44] Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam can converge without any modification on update rules. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 28386–28399. Curran Associates, Inc., 2022. URL [https://proceedings.neurips.cc/paper\\_files/paper/2022/file/b6260ae5566442da053e5ab5d691067a-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/b6260ae5566442da053e5ab5d691067a-Paper-Conference.pdf).

[45] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[46] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[47] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[48] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[49] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *Advances in neural information processing systems*, 32, 2019.

[50] David GT Barrett and Benoit Dherin. Implicit gradient regularization. *arXiv preprint arXiv:2009.11162*, 2020.

[51] Jeff Z HaoChen, Colin Wei, Jason Lee, and Tengyu Ma. Shape matters: Understanding the implicit bias of the noise covariance. In *Conference on Learning Theory*, pages 2315–2357. PMLR, 2021.

[52] Samuel L Smith, Benoit Dherin, David GT Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. *arXiv preprint arXiv:2101.12176*, 2021.

[53] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pages 797–842. PMLR, 2015.

[54] Simon S Du, Chi Jin, Jason D Lee, Michael I Jordan, Aarti Singh, and Barnabas Poczos. Gradient descent can take exponential time to escape saddle points. *Advances in neural information processing systems*, 30, 2017.

[55] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

[56] Jonas Geiping, Micah Goldblum, Phillip E Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. *arXiv preprint arXiv:2109.14119*, 2021.

[57] Nikhil Vyas, Depen Morwani, Rosie Zhao, Gal Kaplun, Sham Kakade, and Boaz Barak. Beyond implicit bias: The insignificance of sgd noise in online learning. *arXiv preprint arXiv:2306.08590*, 2023.

[58] 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.

[59] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[60] Dan Busbridge, Jason Ramapuram, Pierre Ablin, Tatiana Likhomanenko, Eeshan Gunesh Dhekane, Xavier Suau, and Russell Webb. How to scale your EMA. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=DkeeXVdQyu>.

[61] Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 30106–30148. Curran Associates, Inc., 2024. URL [https://proceedings.neurips.cc/paper\\_files/paper/2024/file/350e718ff74062b4bac2c6ffd9e1ac66-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/350e718ff74062b4bac2c6ffd9e1ac66-Paper-Conference.pdf).

[62] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=X6rqEpbnj3>.

[63] Qi Liu and Wanjing Ma. The epochal sawtooth effect: Unveiling training loss oscillations in adam and other optimizers. *arXiv preprint arXiv:2410.10056*, 2024.

[64] Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyang Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, Yangyu Tao, Bin Cui, and Di Wang. Surge phenomenon in optimal learning rate and batch size scaling. *arXiv preprint arXiv:2405.14578*, 2024.

[65] Peter J. Liu, Roman Novak, Jaehoon Lee, Mitchell Wortsman, Lechao Xiao, Katie Everett, Alexander A. Alemi, Mark Kurzeja, Pierre Marcenac, Izzeddin Gur, Simon Kornblith, Kelvin Xu, Gamaleldin Elsayed, Ian Fischer, Jeffrey Pennington, Ben Adlam, and Jascha-Sohl Dickstein. Nanodo: A minimal transformer decoder-only language model implementation in JAX., 2024. URL <http://github.com/google-deepmind/nanodo>.

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

[67] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[68] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

[69] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[70] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[71] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

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

[73] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL <http://distill.pub/2017/momentum>.

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

[75] 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.

[76] Xi Wang and Laurence Aitchison. Batch size invariant adam. *arXiv preprint arXiv:2402.18824*, 2024.

[77] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.

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

[79] Shibo Wang and Pankaj Kanwar. Artur niederfahrenhorst and kourosh hakhamaneshi and rehaan ahmad, 2023. URL <https://www.anyscale.com/blog/fine-tuning-l1lms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2>.

[80] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL [https://datasets-benchmarks-proceedings.neurips.cc/paper\\_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf](https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf).

- [81] Jacob Austin, Sholto Douglas, Roy Frostig, Anselm Levskaya, Charlie Chen, Sharad Vikram, Federico Lebron, Peter Choy, Vinay Ramasesh, Albert Webson, and Reiner Pope. How to scale your model. 2025. Retrieved from <https://jax-ml.github.io/scaling-book/>.
- [82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [83] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th annual international symposium on computer architecture*, pages 1–14, 2023.

## NeurIPS Paper Checklist

### 1. Claims

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

Answer: [Yes]

Justification: We have performed extensive experiments to back our claims. We outline our objectives and follow through with corresponding methods, results, and analysis that substantiate those claims. The abstract offers a concise and truthful summary of the work.

Guidelines:

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

### 2. Limitations

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

Answer: [Yes]

Justification: We discuss the scale limitations of our grid search as we need to perform it on a smaller model compared to [Section 4.4](#) where we perform larger scale experiments but without a grid search. We also clearly state open questions in the discussion section.

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: The results we present are primarily experimental. Although we have looked into results from the theoretical fields, we have not built upon them.

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: We clearly state the models, datasets, and hyperparameters used during training in the paper. We also mention the design choices we adopt, namely, rotatory embeddings, QK-normalization, RMSNorm, GELU activations, separate input and output embeddings, among others. This allows others to replicate the results and validate our main claims.

Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

## 5. Open access to data and code

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

Answer: [\[Yes\]](#)

Justification: Yes, we provide open access to our code, along with clear and sufficient instructions to reproduce our results: <https://github.com/martin-marek/batch-size>.

Guidelines:

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

## 6. Experimental setting/details

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

Answer: [\[Yes\]](#)

Justification: We clearly state the models, datasets, and hyperparameters used during training in the paper. We also mention the design choices we adopt, namely, rotatory embeddings, QK-normalization, RMSNorm, GELU activations, separate input and output embeddings, among others.

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 perform a thorough grid search and report results that reflect the statistical significance of the loss at the optimal hyperparameter settings.

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: Yes, the paper will provide sufficient information on the computational resources used for the experiments—including hardware type, memory, and execution time—in the supplementary materials, which will be included in the appendix. This will ensure reproducibility and support the paper's claims regarding efficiency.

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: Our work fully adheres to the NeurIPS Code of Ethics. We have carefully considered the broader societal and environmental implications of our research and ensured alignment with the conference's ethical standards. Additionally, we affirm our compliance with the guidelines on research integrity, plagiarism, fraud, and reproducibility as outlined in the 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: Yes, the paper discusses the potential positive societal impacts of reducing the resource requirements for training large language models, such as making LLM research more accessible, and enabling training in resource-constrained settings. However, it does not explicitly address potential negative societal impacts, such as the broader consequences of democratizing powerful language technologies without sufficient safeguards.

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: We do not intend to release new datasets or models.

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 have appropriately cited all publicly available and open-source research used in our work. The original authors are explicitly acknowledged and given full credit for their contributions.

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](https://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: We do not intend to release any new models or datasets. However, we intend to make the code publicly available after acceptance and provide proper documentation to the public.

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: Does not apply.

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: Does not apply.

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: Does not apply.

Guidelines:

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

## A Experimental Details

### A.1 Model architecture

Across all of our pretraining experiments, we used four decoder-only transformer models [82] that share the same architecture and only differ in layer dimensions and tokenizer. We specify the exact layer dimensions and tokenizers used in Table 1.

We used a model architecture based on GPT-2 [66] with several small changes based on modern practices to improve model FLOPs utilization (MFU) [43], simplify implementation, and marginally improve model performance:

- Head dimension 128
  - We increased the head dimension on the smaller models from 64 to 128 to achieve a higher MFU. We trained each model on TPU v4, which has  $128 \times 128$  multiply-accumulators. Using a head dimension any lower than 128 would prevent us from filling the multiply-accumulators, thereby lowering MFU. This decision follows the Gemma family of models optimized for TPU training [67].
- RMSNorm [71]
  - We wanted to include the Muon optimizer in our benchmarks, but Muon does not support training 1D layers. To get around this limitation, we simply omitted the use of any 1D layers by using RMSNorm with no bias.
- Untied embeddings
  - We do not tie the weights of the first and last layer. This increases the number of trainable parameters but does not increase the number of active parameters or the cost of training.
- Rotary positional embeddings (RoPE) [69]
- QK-norm [70]
- GELU [72]

We pretrained most models to follow Chinchilla scaling laws, i.e. using 20 tokens per active parameter. The only exception to this rule is the GPT-3 (1.3B) model, which we only trained for 10B tokens, similar to the original GPT-2 (1.5B) model.

Model	30M	19M[14]	GPT-2 (124M)	GPT-3 (1.3B)
Dataset	Fineweb-Edu	C4	Fineweb	Fineweb
Tokenizer	GPT-2	T5	GPT-2	GPT-2
Vocabulary size	50257	32101	50257	50257
Model / embedding dimension	384	512	768	2048
Hidden dimension	1536	2048	3072	8192
Head dimension	128	128	128	128
Number of layers	6	6	12	24
Sequence length	512	512	1024	2048
Non-embedding parameters	11M	19M	85M	1.2B
Embedding parameters	2 x 19M	2 x 16M	2 x 39M	2 x 103M
Active parameters	30M	35M	124M	1.3B
Total trainable parameters	49M	52M	162M	1.4B
Training tokens	600M	705M	2.5B	10B

Table 1: Model dimensions and configurations across different architectures.

### A.2 Learning rate schedule

For pretraining, we used a linear warmup for the first 5% of the training steps (from zero to peak learning rate), followed by cosine decay (from peak learning rate to zero). We use the same schedule for all batch sizes. For fine-tuning, we used a constant learning rate.

### A.3 Stochastic rounding

Across all experiments, we compute activations and store the optimizer state in float32 precision. We also store all trainable model parameters in float32 precision *except* for Adafactor in Figure 8, which uses bfloat16 weights in order to match the memory footprint of LoRA. While it is a common practice to use bfloat16 weights for computation, the master copy of model weights is typically still stored in float32 to enable accumulating small updates to the weights. When the master weights are stored in bfloat16 (which only has  $\sim 2.4$  decimal points of precision), standard deterministic rounding can bias the weight updates, meaning that accumulating gradients across many small batches no longer approximates computing gradients from a single larger batch. Because of this bias, reducing the batch size might actually be detrimental with standard deterministic rounding. One approach to remove this bias is to compute updated weights in float32, then stochastically round them to bfloat16 for storage. In a pretraining experiment in Figure 9, we show that stochastic rounding can bring the performance of bfloat16 weights close to the performance of float32 weights, even for the smallest batch sizes. In the fine-tuning experiment in Figure 8, we did not observe a statistically significant difference between float32 weights and bfloat16 weights with stochastic rounding for Adafactor.

### A.4 Computational cost

We used 32 TPU v4 chips to train all models [83]. Table 2 shows the time for a single training run of each model as well as the total computational cost for each model. Using batch sizes 1 and 2 resulted in roughly a 30-70% drop in MFU compared to using a larger batch size, depending on the model size and sequence length. In practice, we do not recommend using a batch size so small that it severely degrades MFU. We only used the smallest batch sizes to scientifically study the effect of batch size.

Model	30M	19M [14]	GPT-2 (124M)	GPT-3 (1.3B)	Gemma3 (4B)
Accelerator	1 TPU v4 chip	1 TPU v4 chip	1 TPU v4 chip	4 TPU v4 chips	4 TPU v4 chips
Average run time	24 min	25 min	7.5 hours	100 hours	44 min
Num. runs: Figure 1	3110	/	312	11	/
Num. runs: Figure 3	1911	/	/	/	/
Num. runs: Figure 4	6500	/	/	/	/
Num. runs: Figure 5	/	1260	/	/	/
Num. runs: Figure 6	/	/	40	/	/
Num. runs: Figure 8	/	/	/	/	256
Total num. of runs	11521	1260	352	11	256
Total TPU v4 chip hours	4600	525	2640	4400	750

Table 2: Computational cost of each experiment.

## B Additional Results

In Figure 10 we illustrate on a toy problem why momentum is required when the batch size is large but not required when the batch size is small. We run SGD on two variables  $(x, y)$  to minimize the value of a loss function defined as  $x + 10y^2$ . Notice that this loss function is much steeper in the vertical direction compared to the horizontal direction.

We simulate the use of a small and a large batch size by adding noise to the gradient estimate at every step. In particular, we define the minibatch gradient estimate at each step to be the true gradient

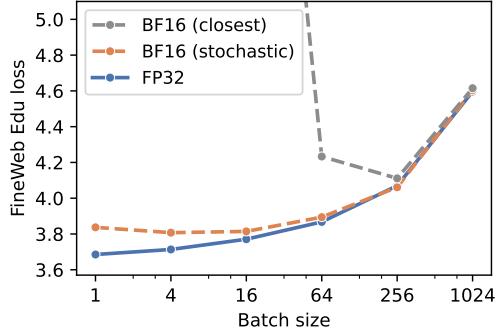


Figure 9: **Stochastic rounding enables bfloat16 training.** We repeat the training runs from Figure 1a using the optimum hyperparameters found for Adafactor. We ablate the precision of the model weights, using: (1) float32 – same as in Figure 1a, (2) bfloat16 with closest rounding, (3) bfloat16 with stochastic rounding. The optimizer state is always stored in float32 precision since it has a negligible size.

multiplied by a random variable sampled from a normal distribution  $\mathcal{N}(1, \sigma^2)$ , where the noise scale  $\sigma$  controls the signal-to-noise ratio of the minibatch gradient estimate. We simulate a large batch size by using a high signal-to-ratio of 5, and a small batch size by using a small signal-to-noise ratio of only 0.3.

In our experiments training language models, we fixed the computational budget across all batch sizes by only training for a single epoch. Analogously in this toy example, we only run 10 optimization steps using the simulated large batch size, but we run 100 optimization steps using the small batch size. The idea is that if we use a small batch size, the minibatch gradient estimates at each step become more noisy, but in turn we get to take more optimizer steps.

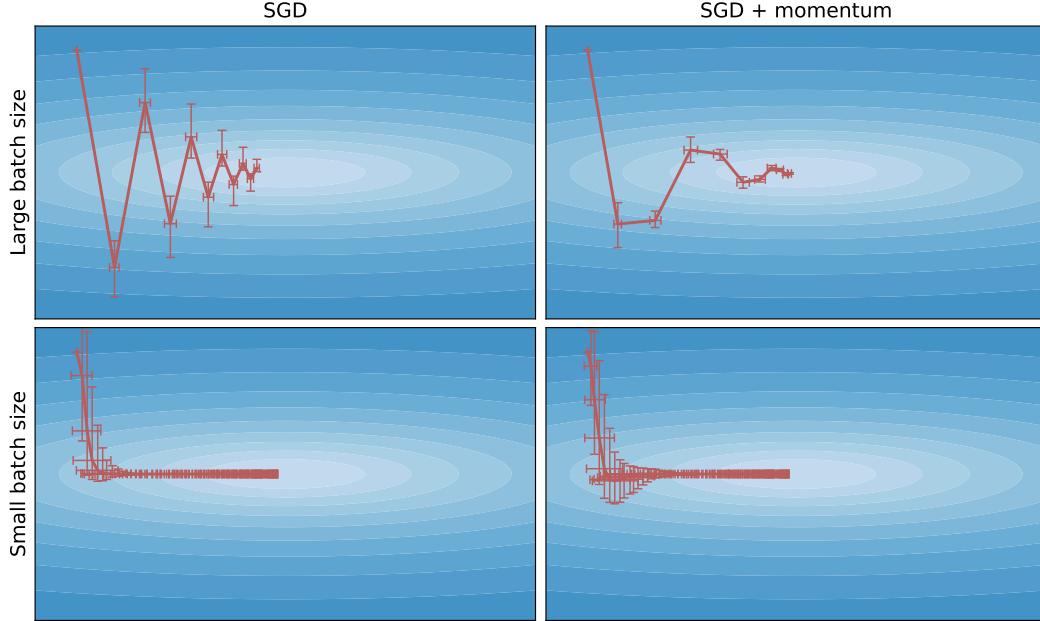


Figure 10: **Toy optimization problem.** We compare the use of SGD and SGD with momentum on a loss function defined as  $x + 10y^2$ . We simulate a small and a large batch size by adding Gaussian noise to the gradient estimate at each optimizer step. The error bars show a 50% interquartile range sampled across different random seeds for the minibatch gradient noise.

In the top row of Figure 10, we see that when a large batch size is used, we only get to take a small number of steps, and so in turn every step has to be large. Because the loss function is much steeper in the vertical direction compared to the horizontal direction, the large step size results in oscillations along the vertical direction. In this case, using momentum helps dampen the oscillations along the vertical direction and speeds up convergence along the horizontal direction. In fact, in the extreme case of full-batch gradient descent (which has no gradient noise), using momentum probably improves the convergence rate of gradient descent for this toy problem [73].

We show the small batch size setting in the bottom row of Figure 10. Notice that since the batch size is small, every step becomes more noisy, but in turn we can take more steps and reduce the size of each step. As a result of decreasing the step size, the optimizer no longer overshoots along the vertical direction, obviating the need for momentum. There are no longer any oscillations to dampen, so using SGD with and without momentum results in similar performance.