Adaptive Batch Size Schedules for Distributed Training of Language Models with Data and Model Parallelism

Tim Tsz-Kit Lau[*†‡] Weijian Li[*§] Chenwei Xu[§] Han Liu[§] Mladen Kolar[†¶]
[‡]University of Pennsylvania [§]Northwestern University [¶]University of Southern California
timlautk@upenn.edu; {weijianli2021,chenweixu2023}@u.northwestern.edu;
hanliu@northwestern.edu; mkolar@marshall.usc.edu

An appropriate choice of batch sizes in large-scale model training is crucial, yet it involves an intrinsic yet inevitable dilemma: large-batch training improves training efficiency in terms of memory utilization, while generalization performance often deteriorates due to small amounts of gradient noise. Despite this dilemma, the common practice of choosing batch sizes in language model training often prioritizes training efficiency—employing either constant large sizes with data parallelism or implementing batch size warmup schedules. However, such batch size schedule designs remain heuristic and often fail to adapt to training dynamics, presenting the challenge of designing adaptive batch size schedules. Given the abundance of available datasets and the data-hungry nature of language models, data parallelism has become an indispensable distributed training paradigm, enabling the use of larger batch sizes for gradient computation. However, vanilla data parallelism requires replicas of model parameters, gradients, and optimizer states at each worker, which prohibits training larger models with billions of parameters. To optimize memory usage, more advanced parallelism strategies must be employed. In this work, we propose general-purpose and theoretically principled adaptive batch size schedules compatible with data parallelism and model parallelism. We develop a practical implementation with PyTorch Fully Sharded Data Parallel, facilitating the pretraining of language models of different sizes. We empirically demonstrate that our proposed approaches outperform constant batch sizes and heuristic batch size warmup schedules in the pretraining of models in the Llama 2 family, with particular focus on smaller models with up to 3 billion parameters. We also establish theoretical convergence guarantees for such adaptive batch size schedules with AdaM for general smooth nonconvex objectives.

1. Introduction

Large-batch training (i.e., using large batch sizes) is arguably the current *de facto* training paradigm for large language models, driven by recent advances and the availability of computational hardware for deep learning. For instance, the open-weight model, Llama 3 405B [1], utilizes a batch size of 1024 sequences of length 4096, resulting in 4M tokens per batch. Despite the efficient utilization of available hardware through parallelization, a major drawback of large-batch training is the issue of "generalization gap" (see e.g., [2])—where model generalization performance deteriorates compared to small-batch training without heavy tuning of other hyperparameters. See Figure 1 for a graphical illustration of the existence of generalization gaps with different batch sizes when training a vanilla transformer with 61M parameters. Keskar et al. [3] argued that small-batch methods tend to converge to flat minima, leading to better generalization. To close this generalization gap, several works [4–6] have proposed using large learning rates to offset the effect of large batch sizes, recovering the

[*]Equal contribution

[†]Part of the work of Tim Tsz-Kit Lau and Mladen Kolar was performed when they were at The University of Chicago Booth School of Business.

Second Conference on Parsimony and Learning (CPAL 2025).

generalization performance of using small batches. However, the training of language (and vision) models based on the attention mechanism [7] and the transformer architecture is notoriously unstable. Reducing training instability, including unwanted loss spikes (see e.g., [8, 9]), demands significant tuning and cautious hyperparameter selections, like using a small learning rate.

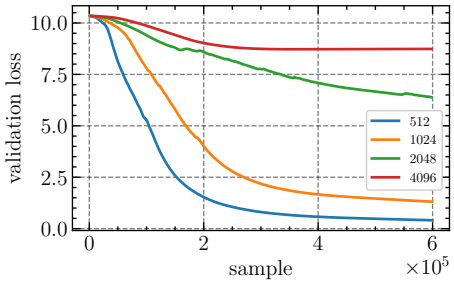

Figure 1: Generalization gap in transformer pretraining.

Beyond using a large learning rate to balance the intrinsic trade-off between training efficiency and generalization performance of large-batch training, Keskar et al. [3] also suggested the use of *adaptive sampling methods* [10, 11]. These methods are essentially adaptive batch size schemes that progressively improve the accuracy of the batch gradient approximation by gradually increasing batch sizes throughout the model training process. This concept has been explored by De et al. [12, 13] and Lau et al. [14], but their implementations are limited to the single-device setting, where all data samples are implicitly assumed to reside on the same device. This limitation makes them unfit for data-parallel distributed training wherein data is spread across various workers in a parallel system, potentially encompassing several network-connected nodes, thereby preventing the scaling necessary to train large models. Beyond the single-device setting, Lau et al. [15] have also extended such adaptive batch size schemes to local gradient methods for local batch sizes, where model synchronization is performed every several gradient steps rather than every step.

Data parallelism [16], such as `DistributedDataParallel` (`DDP`) in PyTorch [17] and counterparts in TensorFlow [18] and JAX [19, 20], is arguably the most popular paradigm for distributed training in deep learning. In data parallelism (alone), each worker holds a local copy of the model parameters (as well as gradient and optimizer states). The global input batch is divided into multiple minibatches for each training step, so each worker performs forward and backward computations with a different minibatch. After each training step, all GPUs perform an *all-reduce* collective communication to synchronize gradients, followed by a global model parameter update. This ensures that all local copies of the model remain identical after the parameter update steps. Adaptive batch size schemes can be developed based on the approaches in [10, 11, 21] for data-parallel settings, providing practical adaptive batch size schedules in PyTorch `DDP` for training large-scale deep neural networks, which require data parallelism.

While these practical schemes open up the possibility of distributed training of larger models with GPUs of lower memory, they are constrained by the inherent design of `DDP`—the need to maintain a model replica at each worker. State-of-the-art large language models (LLMs) now consist of billions or even hundreds of billions of parameters (e.g., Llama 3 405B [1]). Distributed training with only data parallelism thus unfortunately fails, as the memory required to store such models well exceeds the available memory of a single GPU. Even worse, access to expensive workstation-level GPUs with more memory is often limited to industrial labs, whereas academic researchers and end-users often have to resort to less powerful consumer-level GPUs or workstation-level GPUs with less memory.

To alleviate this limitation inherent to data parallelism, more memory-efficient paradigms of parallelism, such as model parallelism [22], have been proposed. In model parallelism, model parameters are sharded into various components and distributed to different workers. In particular, PyTorch Fully Sharded Data Parallel (`FSDP`) [23] is an implementation of model parallelism in PyTorch [24], marking the first native feature in PyTorch that can support models with up to trillions of parameters without relying on more sophisticated third-party libraries for model parallelism such as DeepSpeed [25], Megatron-LM [22, 26, 27], and their combinations [28], which could be overwhelming to get started with and too technical to modify for users' specific needs. Moreover, PyTorch `FSDP` has been widely adopted in the pretraining of various open-source language models such as OPT [29], TinyLlama [30], OLMo [31, 32], and DRBX [33].

However, even with data parallelism and model parallelism, LLM pretraining involving models with up to hundreds of billions of parameters and trillions of tokens (e.g., Llama 3 405B [1]), still incurs extensive costs (more than millions of US dollars per model) and imposes a significant carbon footprint. Consequently, there is a pressing need for developing proper and well-crafted training strategies. In this work, we focus on choosing dynamic batch size schedules, which deserve more attention than they have, since, unlike other optimizer hyperparameters, batch sizes also control training efficiency via memory utilization of GPUs, in addition to affecting model generalization performance and training stability. The current practice of choosing batch sizes in LLM pretraining, however, remains heuristic, in the sense that it usually involves either constant large batch sizes or prespecified heuristic warmup schedules which could be very hard to design.

Contributions. In this work, we propose theoretically principled adaptive batch size schedules based on the adaptive sampling method [10] for pretraining large language models, which are also generally applicable to training other deep neural networks. On the theoretical front, we establish a convergence guarantee for the proposed adaptive batch size schedules for ADAM, the *de facto* optimizer for pretraining language models. Various recent works have shown, both empirically and theoretically, that ADAM outperforms SGD in training attention-based language models [34–37]. Our convergence guarantee complements the existing results of adaptive batch size schedules for SGD [12, 13] and ADAGRAD [14]. From a practical perspective, we develop a solution of adaptive batch size schedules based on PyTorch `FSDP`, which are tailor-made for pretraining LLMs with more than billions of parameters.

2. Related Work

Large-batch training of language models. Large-batch training has proven to be very successful for different deep learning applications including computer vision [38, 39] and natural language processing [40–42]. From an empirical perspective, many open-source or open-weights models, such as OPT [29], BLOOM [43], Mistral 7B [44], Baichuan 2 [45], Qwen [46, 47], OLMo [31, 32], Gemma [48, 49], Llama [1, 50] and DeepSeek [51, 52], revealed that they were pretrained with large numbers of GPUs or TPUs (i.e., data-parallel sizes), hence naturally making use of large-batch training. While using large batch sizes is now standard, the rationale for choosing the magnitude of such large batch sizes is mostly based on hardware availability. Only recently in the training of Stable LM 2 1.6B, Bellagente et al. [53] clarified the selection of global batch sizes, aiming to strike an optimal balance between minimizing training time and the extra training tokens needed to reach the desired final training loss. Shallue et al. [54] study the effects of data parallelism by performing ablation studies on different batch sizes by training different models on different datasets using different optimizers, finding no evidence that large batch sizes degrade generalization performance with careful hyperparameter search. From a more theoretical perspective, McCandlish et al. [55] develop a model for understanding the *critical batch size* that determines the tradeoff between speed and efficiency of large-batch training. Kaplan et al. [56] further study the *scaling law* of the critical batch size as a power of the training loss only for language models. However, in most of these works, benchmarking was performed with different magnitudes of constant batch sizes, with the notable exception of McCandlish et al. [55] which provided a case study of dynamically varying the batch size with an adaptive batch size schedule, but only using a simple model (CNN) and dataset (SVHN). The effect of adaptive batch sizes for pretraining language models, to the best of our knowledge, remains elusive to the community.

Batch size schedules. Adaptive sampling methods [10, 11, 21], which adjust batch sizes based on gradient noise or gradient approximation quality, are further explored in deep learning [12–14, 57] but have not been applied to data parallelism with distributed samplers. The development of adaptive batch size schedules for deep learning is not a novel concept, featuring methodologies such as Big Batch SGD [12, 13], CABS [58], AdaBatch [59], SimiGrad [60] and AdaScale SGD [61]. Our work is also closely related to and motivated by the heuristic technique of batch size warmup/batch ramp, which has been widely adopted in pretraining LLMs and even in reinforcement learning [62]. Batch size warmup usually involves prespecified schedules of multiple batch size stages starting

from training with multiple increasing smaller batch sizes for small portions of the total training tokens, followed by training with the remaining tokens using a large batch size. For instance, GPT-3 [63] was pretrained by gradually increasing the batch size linearly from a small value (32k tokens) to the full value (3.2M tokens) over the first 4–12 billion tokens of training. Nemotron-4 [64] was pretrained with a batch size schedule of batch sizes 384–768–1152 sequences for 2.5%–2.5%–95% of the total number of training tokens. Llama 3 405B [1] was trained using the following batch size schedule: an initial batch size of 4M tokens with a sequence length 4096 tokens for 252M tokens; a batch size of 8M tokens with a sequence length of 8192 tokens for 2.87T tokens; a batch size of 16M tokens for the remainder of a total of about 15T training tokens. Such a batch size recipe is found to be able to stabilize training—few loss spikes were observed and it did not require interventions to correct for model training divergence. Despite potentially improving training efficiency or data parallelism, batch size warmup schedules remain heuristic and their impact on training is difficult to grasp. Another related yet seemingly orthogonal technique is sequence length warmup [65, 66], which progressively grows the sequence length throughout the pretraining process. Note that the pretraining of Llama 3 405B employs both batch size warmup and sequence length warmup.

3. Adaptive Batch Size Schedules with 2D Parallelism

We present the adaptive batch size schedules for data and model parallelism (termed *2D parallelism*), facilitating the scaling of pretraining for models with billions of parameters.

Notation. We define $[\![n]\!] \coloneqq \{1, \dots, n\}$ for $n \in \mathbb{N}^* \coloneqq \mathbb{N} \setminus \{0\}$. We denote the inner product in $\mathbb{R}^d$ by $\langle \cdot, \cdot \rangle$ and its induced L_2-norm by $\|\cdot\|$, and $\|\cdot\|_1$ stands for the L_1-norm. For a vector $x \in \mathbb{R}^d$, $[x]_j$ denotes its jth coordinate ($j \in [\![d]\!]$). For a function $f \colon \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$, $\partial_j f$ denotes its partial derivative with respect to its jth coordinate for $j \in [\![d]\!]$. The ceiling function is denoted by $\lceil \cdot \rceil$. The disjoint union of sets $\mathcal{S}_1, \dots, \mathcal{S}_J$ is denoted by $\bigsqcup_{j \in [\![J]\!]} \mathcal{S}_j$.

3.1. Vanilla Adaptive Batch Size Schedules

We consider the empirical risk minimization problem in which we want to minimize the loss function $\mathscr{L} \colon \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$ in the form of a finite-sum objective:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \ \mathscr{L}(w) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \ell(w; z_i), \tag{1}$$

where $\ell \colon \mathbb{R}^d \times \mathcal{Z} \to \mathbb{R} \cup \{\pm\infty\}$ is the individual loss function, and $\mathcal{D}_n \coloneqq \{z_i\}_{i=1}^n$ is the set of n training samples. If $\ell(\cdot; z)$ is continuously differentiable for any $z \in \mathcal{Z}$, then the gradient of the loss function and its batch counterpart (i.e., the batch gradient) are given by

$$\nabla \mathscr{L}(w) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(w; z_i) \quad \text{and} \quad \nabla \mathscr{L}_{\mathcal{B}}(w) \coloneqq \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla \ell(w; z_i),$$

where the batch $\mathcal{B} \subseteq [\![n]\!]$ is a subset of indices of data points sampled uniformly without replacement, and $b \coloneqq |\mathcal{B}|$ is the corresponding batch size. We write $\ell_i(w) \coloneqq \ell(w; z_i)$ and $\nabla \ell_i(w) \coloneqq \nabla \ell(w; z_i)$. The batch gradient $\nabla \mathscr{L}_{\mathcal{B}}$ is used to approximate the full gradient $\nabla \mathscr{L}$ as the number of samples n is prohibitively large.

Norm test. Falling into the family of adaptive sampling methods, the *norm test* [10] is motivated by measuring the quality of the approximation of the full gradient $\nabla \mathscr{L}$ by the batch gradient $\nabla \mathscr{L}_{\mathcal{B}}$ through the lens of approximation of a descent direction for the loss $\mathscr{L}$. If $\ell(\cdot; z)$ is also convex, then $-\nabla \mathscr{L}_{\mathcal{B}}$ is a descent direction for $\mathscr{L}$ at $w \in \mathbb{R}^d$ if and only if $\langle \nabla \mathscr{L}_{\mathcal{B}}(w), \nabla \mathscr{L}(w) \rangle \geqslant 0$, that is, $\mathscr{L}_{\mathcal{B}}$ and $\mathscr{L}$ share the same direction at w. It can be shown that the above inner product condition is equivalent to the norm condition: $\|\nabla \mathscr{L}_{\mathcal{B}}(w) - \nabla \mathscr{L}(w)\| \leqslant \eta \|\nabla \mathscr{L}(w)\|$ for any $\eta \in [0, 1)$. This condition cannot be checked directly, since the number of samples n is in billions for LLMs and the full gradient $\nabla \mathscr{L}$ is unavailable. Instead, we have to resort to a batch approximation:

$$\frac{\|\mathrm{Var}_{i \in \mathcal{B}}(\nabla \ell_i(w))\|_1}{b} \cdot \frac{n - b}{n - 1} \leqslant \eta^2 \|\nabla \mathscr{L}_{\mathcal{B}}(w)\|^2, \tag{2}$$

where

$$\mathrm{Var}_{i\in\mathcal{B}}(\nabla\ell_i(w)) \coloneqq \frac{1}{b-1}\sum_{i\in\mathcal{B}}(\nabla\ell_i(w)-\nabla\mathfrak{L}_{\mathcal{B}}(w))^2.$$

The adjustment factor $(n-b)/(n-1)$ is approximated by 1 as we take $n\to\infty$. Consequently, to ensure that the batch gradient approximates the descent direction of the full objective $\mathfrak{L}$ well, the (*approximate*) *norm test* checks the following condition at each iteration $k\in\mathbb{N}^*$:

$$\frac{\|\mathrm{Var}_{i\in\mathcal{B}_k}(\nabla\ell_i(w_k))\|_1}{b_k} = \frac{1}{b_k(b_k-1)}\sum_{i\in\mathcal{B}_k}\left[\|\nabla\ell_i(w_k)-\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\|^2\right] \leqslant \eta^2\|\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\|^2, \quad (3)$$

and increases the next batch size b_{k+1} if the above inequality is not satisfied, using

$$b_{k+1} = \left\lceil \frac{\|\mathrm{Var}_{i\in\mathcal{B}_k}(\nabla\ell_i(w_k))\|_1}{\eta^2\|\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\|^2}\right\rceil.$$

The condition can be viewed as an approximation of the following *exact variance norm test* in the stochastic setting:

$$\mathbb{E}_k\left[\|\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)-\nabla\mathfrak{L}(w_k)\|^2\right] \leqslant \eta^2\|\nabla\mathfrak{L}(w_k)\|^2, \quad (4)$$

i.e., the motivating norm condition holds in expectation. Here $\mathbb{E}_k \coloneqq \mathbb{E}[\cdot\,|\,\mathcal{F}_k]$ denotes the conditional expectation with respect to the σ-algebra up to the current batch at iteration k, i.e., $\mathcal{F}_k \coloneqq \sigma(\{w_0,\mathcal{B}_0,\mathcal{B}_1,\ldots,\mathcal{B}_{k-1}\})$. After the next batch size is determined, the training loop continues with an optimizer step. The test implicitly makes a heuristic assumption that the next batch of size b_{k+1} will satisfy the approximate norm test at the current iterate w_k, but this is never checked to streamline the training loop.

3.2. Adaptive Batch Size Schedules with Data Parallelism

To allow training with large batch sizes with parallelized computations, a data-parallel extension of the *norm test*, which is referred to as DDP-Norm, can be developed and can be implemented based on PyTorch `DDP`. A special treatment of the norm test with data parallelism is necessary since data samples now reside in different workers, but we need to compute the mean and the variance of all the per-sample gradients in the norm test.

Specifically, at each iteration k, the global batch $\mathcal{B}_k$ is split across J workers with minibatches $(\mathcal{B}_{k,j})_{j\in[\![J]\!]}$ of equal size $b_{k,J}$ such that the global batch is the disjoint union of all minibatches, i.e., $\mathcal{B}_k = \bigsqcup_{j\in[\![J]\!]}\mathcal{B}_{k,j}$. Notice that at each worker $j\in[\![J]\!]$, the minibatch gradient can be computed by $\nabla\mathfrak{L}_{\mathcal{B}_{k,j}}(w_k) = {}^1\!/_{b_{k,J}}\sum_{i\in\mathcal{B}_{k,j}}\nabla\ell_i(w_k)$. Since the minibatches have equal size and are disjoint, applying the law of total expectation, the global batch gradient is equal to $\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k) = {}^1\!/_J\sum_{j=1}^J\nabla\mathfrak{L}_{\mathcal{B}_{k,j}}(w_k)$. Note that the averages across workers are computed using *all-reduce* operations in PyTorch `DDP`. When minibatch sizes exceed the maximum memory of the workers, the technique of gradient accumulation is applied to simulate larger global batch sizes.

It is worth noting that efficiently implementing the approximate norm test (3) in deep learning libraries such as PyTorch [24] is highly nontrivial, since per-sample gradients $\nabla\ell_i(w_k)$ are unavailable in the backward step of a standard training loop, but only the batch gradient $\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)$ under a single-device setting or the minibatch gradient $\nabla\mathfrak{L}_{\mathcal{B}_{k,j}}(w_k)$ at each worker j under PyTorch `DDP`. If we were to implement the native approximate norm test (3), we would have had to compute per-sample gradients in parallel using vectorized mappings and based on a deep copy of the model, leading to undesirable memory and computational overheads. Thus, in practical implementation under data parallelism, instead of the approximate norm test (3), we propose to make use of the minibatch gradients of the workers to construct an estimator for the gradient variance

$$\widehat{\mathrm{Var}}_{i\in\mathcal{B}_k}(\nabla\ell_i(w_k)) \coloneqq \frac{1}{J}\sum_{j\in[\![J]\!]}\left(\nabla\mathfrak{L}_{\mathcal{B}_{k,j}}(w_k)-\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\right)^2,$$

leading to the following more efficient implementation:

$$\frac{1}{b_k}\cdot\frac{1}{J}\sum_{j\in[\![J]\!]}\left[\left\|\nabla\mathfrak{L}_{\mathcal{B}_{k,j}}(w_k)-\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\right\|^2\right] \leqslant \eta^2\|\nabla\mathfrak{L}_{\mathcal{B}_k}(w_k)\|^2. \quad (5)$$

From now on, we refer the above alternative test as DDP-Norm. This implementation is much more computationally efficient since the minibatch gradients $\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k)$ are already available at each worker and the global batch gradient $\nabla \mathscr{L}_{\mathcal{B}_k}(w_k)$ can be computed using *all-reduce* operations. Note however that this implementation requires an additional *all-reduce* operation every time to compute the quantity on the left hand side of (5) and additional memory to store it.

3.3. Adaptive Batch Size Schedules with 2D Parallelism via PyTorch FSDP

To enable the training of models with more than billions of parameters, model-parallel training presents a more sophisticated paradigm of parallelism. It shards the parameters of models and allocates different shards to different workers. In essence, PyTorch FSDP [23], which shares similarities with ZeRO-3 [67, 68] in DeepSpeed [25], operates by substituting the *all-reduce* operation in PyTorch DDP with *all-gather* and *reduce-scatter* operations.

For the purpose of mathematical illustration, we focus particularly on the tensor parallelism aspect of model parallelism. Coupled with data parallelism, it is established that each worker j possesses its own set of sharded parameters $\mathcal{W}_j$, $j \in [\![J]\!]$, such that all the model parameters are denoted by $w_k = (w_{k,j})_{j \in [\![J]\!]}$. Here, the sharded parameters on worker j are represented by $w_{k,j} \in \mathcal{W}_j$. Consequently, to compute the microbatch gradient at worker j, the gradients of all parameter shards must be resharded to obtain $\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k) = (\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_{k,1}), \dots, \nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_{k,J}))$, which can be efficiently implemented using the API of PyTorch FSDP. The implementation of DDP-Norm based on PyTorch FSDP is referred to as FSDP-Norm.

4. Convergence Analysis

Complementary to the convergence results of the norm test for SGD [12, 13] and AdaGrad [14], we derive convergence guarantees for Adam, acknowledging its prevalence in training deep neural networks for both computer vision and, more recently, language models. Adam [69] employs the following update formula (with bias corrections for m_k and v_k dropped):

$$(\forall k \in \mathbb{N}^*) \quad m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k, \quad v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2, \quad w_{k+1} = w_k - \alpha m_k \odot v_k^{-1/2}, \tag{6}$$

where $g_k \coloneqq \nabla \mathscr{L}_{\mathcal{B}_k}(w_k)$, $\alpha > 0$ is a constant learning rate, $(m_k)_{k \in \mathbb{N}^*}$ and $(v_k)_{k \in \mathbb{N}^*}$ are the sequences of exponential weighted moving averages of the first two moments of the batch gradients respectively, $(\beta_1, \beta_2) \in (0, \infty)^2$ are weighting parameters, $\odot$ denotes the Hadamard product, and the power operations are performed coordinate-wise. We omit the bias corrections of m_k and v_k to simplify the analysis, but note that it can be easily extended to incorporate bias corrections. We also consider the more challenging scenario where $v_k^{-1/2}$ instead of $(v_k^{1/2} + \varepsilon)^{-1}$ is used in the update, since the denominator of the adaptive step sizes is no longer lower bounded away from 0. In our analysis, we invoke the following assumptions.

Assumption 1 (L-Lipschitz smoothness)**.** The loss function $\mathscr{L}$ is L-Lipschitz smooth ($L > 0$): for any $(u, v) \in \mathbb{R}^d \times \mathbb{R}^d$, we have $\|\nabla \mathscr{L}(u) - \nabla \mathscr{L}(v)\| \leqslant L \|u - v\|$.

Similarly to the analysis for AdaGrad [14], we also require a coordinate-wise version of the (exact variance) norm test to hold due to the use of adaptive step size.

Proposition 1. *The* coordinate-wise (*exact variance*) *norm test with constant $\eta \in (0, 1)$ ensures that, for every iteration $k \in [\![K]\!]$, the coordinate-wise batch gradient $\partial_i \mathscr{L}_{\mathcal{B}_k}(w_k)$ satisfies the following* coordinate-wise expected strong growth (*E-SG*) *condition: for all $i \in [\![d]\!]$, we have*

$$\mathbb{E}_k[(\partial_i \mathscr{L}_{\mathcal{B}_k}(w_k))^2] \leqslant (1 + \eta^2)(\partial_i \mathscr{L}(w_k))^2.$$

Following closely a similar analysis to that in [70], we provide the following convergence results of the norm test for Adam.

Theorem 1. *Suppose that Assumption 1 holds. Let $(w_k)_{k \in \mathbb{N}^*}$ be the Adam iterates generated by (6), where the batch size $b_k \coloneqq |\mathcal{B}_k|$ is chosen such that the coordinate-wise (exact variance) norm test with constant*

$\eta \in (0,1)$ *is satisfied at each iteration* $k \in \mathbb{N}^*$. *Then, if* $0 < \beta_1 \leqslant \sqrt{\beta_2} - 8(1+\eta^2)(1-\beta_2)/\beta_2^2$ *and* $\beta_2 \in (0,1)$, *we have* $\sum_{k=1}^{K} \mathbb{E}[\|\nabla \mathscr{L}(w_k)\|] \leqslant \widetilde{\mathcal{O}}(K)$, *where* $\widetilde{\mathcal{O}}$ *hides any logarithmic factors.*

The full statement of this theorem and its proofs, as well as more in-depth related discussions, are deferred to Appendix B. The convergence results presented do not account for the decoupled weight decay in AdamW [71], which is more commonly used as an optimizer for language model pretraining. Furthermore, considerations such as learning rate schedules and gradient clipping are not included in these findings. Extending the above convergence guarantees to these settings is highly challenging and nontrivial and is left for future work.

5. Numerical Experiments

To showcase the versatility and scalability of FSDP-Norm, we conduct experiments with various families of decoder-only autoregressive language models at with different sizes and pretraining datasets. These include MicroLlama 300M [72], TinyLlama 1.1B [30] and OpenLlama 3B [73] on the C4 dataset [74]. The C4 dataset are tokenized using the Llama 2 tokenizer [50] with a vocabulary size of 32,000. Experiments are conducted on workstations equipped with 4 NVIDIA L40S GPUs (MicroLlama) and 4 NVIDIA A100-SXM 80GB GPUs (TinyLlama and OpenLlama). The training of the latter two models only feasible with PyTorch `FSDP` but not with PyTorch `DDP` using such hardware configurations, even with mixed-precision training (`bfloat16` is used). Our implementation utilizes the PyTorch `FSDP` API in PyTorch 2.6.1 and is simplified through Lightning Fabric of Lightning 2.4 [75]. For the ease of training language models, we also use LitGPT 0.5.3 [76]. Open-source implementation of DDP-Norm and FSDP-Norm is available at `https://github.com/timlautk/adaptive-batch-fsdp`.

Training Specifications. Adhering to the pretraining configurations of open-source LLMs such as TinyLlama [30] and OLMo [31], our training specifications include a linear warmup followed by a cosine decay learning rate schedule, and the AdamW optimizer with weight decay and gradient clipping. The adaptive batch size schedule is set to a maximum global batch size, above which the norm test is no longer performed, opting for fixed interval testing over step-by-step (a test interval 1 is used, but longer interval entails reduced overheads brought by the test). Efficiency dictates using the test in its original form rather than its coordinate-wise variant, despite convergence guarantees. Given that batch sizes increase to the maximum possible values in the early stages, we only pretrain our models for a number of samples that are sufficient to display the behavior of our method, treating these experiments mainly as proofs of concept. Detailed configurations are provided in Appendix C.

5.1. MicroLlama 300M

We first pretrain MicroLlama with 300M trainable parameters on the C4 dataset [74] under the same sets of other hyperparameters in order to better understand the effect of adaptive batch sizes. We compare with various constant batch size baselines $b_k \in \{2048, 4096, 8192\}$ and a stagewise batch size schedule 2048-4096-8192 for 2.5-2.5-95% of training tokens mimicking a popular batch size warmup for pretraining LLMs, and plot the results in Figure 2. We apply DDP-Norm for this relatively small model to demonstrate the applicability of the proposed schedules with PyTorch `DDP`. In Table 1, we report the total number of gradient steps (step), average batch size (bsz.), wall-clock time (time; in hours), best training loss (loss) and best validation loss (val loss; estimated by 100 iterations).

We observe from Figure 2 that with $\eta = 0.2$ or $\eta = 0.275$, our proposed DDP-Norm outperforms the constant batch size baselines by a large margin in terms of validation loss. Specifically, using the same number of training samples, from Table 1, our method achieves lower validation losses when using similar number of steps ($\eta = 0.2$ versus $b_k = 8192$), when we use the number of steps as the criterion of measuring training efficiency. Our proposed schedule with $\eta = 0.2$ performs slightly worse than the stagewise batch size schedule, but it is expected since the latter has a smaller averaged batch size and takes a larger number of training steps. It is also worth noting that the design of the stagewise schedule is completely heuristic and might require lots of tuning, e.g., the number of stages, values of batch sizes and their ratios.

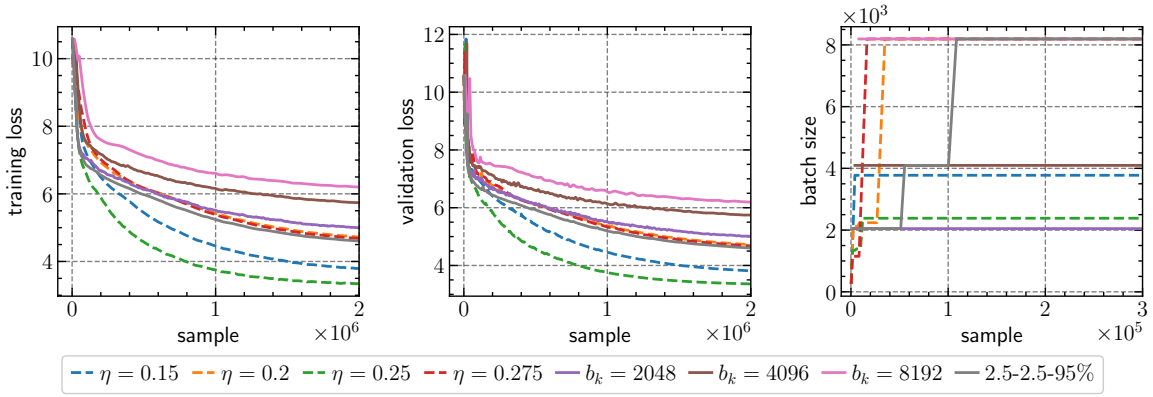

Figure 2: Training loss, validation loss and batch size schedule for MicroLlama 300M

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.15$	531	3770	9.31	3.764	3.811
$\eta = 0.2$	254	7878	8.85	4.699	4.720
$\eta = 0.25$	843	2373	10.30	3.313	3.361
$\eta = 0.275$	252	7965	8.84	4.669	4.677
$b_k = 2048$	977	2048	11.18	4.976	5.005
$b_k = 4096$	489	4096	9.66	5.722	5.741
$b_k = 8192$	245	8192	8.48	6.183	6.192
2.5-2.5-95%	269	7439	8.78	4.594	4.604

Table 1: Results of MicroLlama 300M

We also observe that our method uses smaller batches at early stages and larger batches at later stages of training (e.g., $\eta \in \{0.2, 0.275\}$). This behavior has greater benefits regarding training efficiency because a larger batch size at each step means fewer number of required steps for the whole training process. On the other hand, our method greatly mitigates the side-effect of large-batch training—higher validation loss at the end of training—by starting from a small batch size and adaptively increasing it. Thus, our method enjoys both the good generalization performance of small batches and the high training efficiency of large batches. More importantly, our method is able to automatically increase batch sizes whenever it is necessary, to values that are completely adaptive to the training dynamics. Taking the adaptive batch size schedules in Figure 2 as an example, it is almost impossible to hand-craft similar schemes.

5.2. TinyLlama 1.1B

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.05$	261	7676	32.53	5.663	5.671
$\eta = 0.075$	267	7521	32.67	5.705	5.704
$\eta = 0.08$	270	7415	32.61	5.109	5.113
$\eta = 0.085$	274	7312	32.83	4.257	4.256
$b_k = 4096$	489	4096	34.48	3.814	3.817
$b_k = 8192$	245	8192	32.41	4.895	4.893
2.5-2.5-95%	269	7439	32.80	4.368	4.367

Table 2: Results of TinyLlama 1.1B

We also pretrain TinyLlama 1.1B on the C4 dataset, which necessitates the use of PyTorch `FSDP` and FSDP-Norm. From Figure 3 and Table 2, similar conclusions can be made. We observe that our proposed FSDP-Norm effectively narrows the generalization gap between large and small batches, compared with constant batch sizes and with stagewise batch size schedule baselines. Specifically, our method facilitates the adoption of larger batch sizes of 8192 during the later stages of training. For instance, our method with $\eta = 0.085$ achieves an averaged batch size of 7312, yet it achieves validation loss closer to that of $b_k = 4096$, compared to $b_k = 8192$. Our proposed method is also able to reduce the magnitude of potential loss spikes which are obvious in using constant batch sizes.

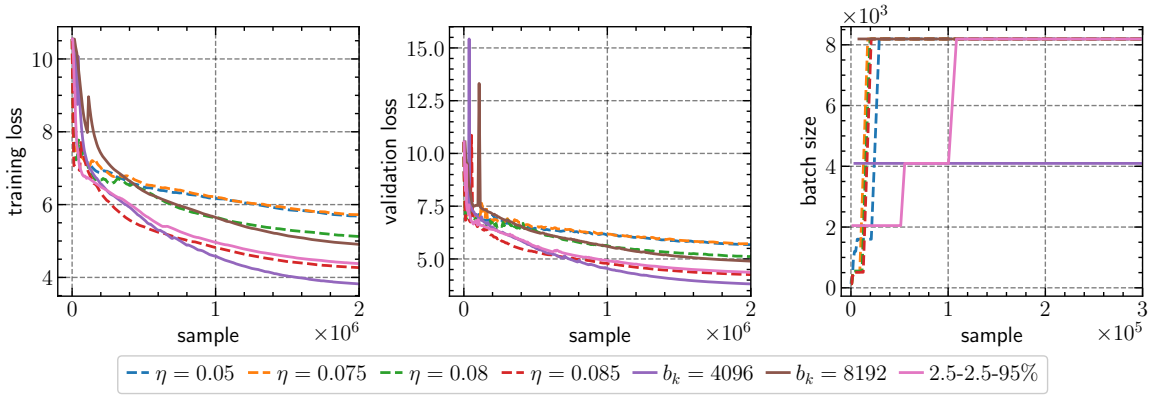

Figure 3: Training loss, validation loss and batch size schedule for TinyLlama 1.1B

5.3. OpenLlama 3B

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.05$	249	8045	19.54	4.943	4.935
$\eta = 0.1$	253	7926	19.73	5.026	5.031
$\eta = 0.15$	259	7726	19.59	4.549	4.554
$b_k = 4096$	489	4096	20.75	3.934	3.956
$b_k = 8192$	245	8192	19.53	5.113	5.104
2.5-2.5-95%	269	7439	19.59	4.776	4.781

Table 3: Results of OpenLlama 3B

We finally pretrain OpenLlama 3B on the C4 dataset, where a shorter sequence length of 512 instead of 2048 is used due to constraint on compute resources. Again, we observe similar phenomena to those of the smaller models, as revealed in Figure 4 and Table 3. Specifically, with $\eta = 0.15$, the proposed approach requires slightly longer training time and larger number of training steps than the constant batch size 8192, while achieving a lower validation loss. While using a constant batch size 4096 achieves an even lower validation loss, it requires substantially more training steps and more than one hour of additional training time.

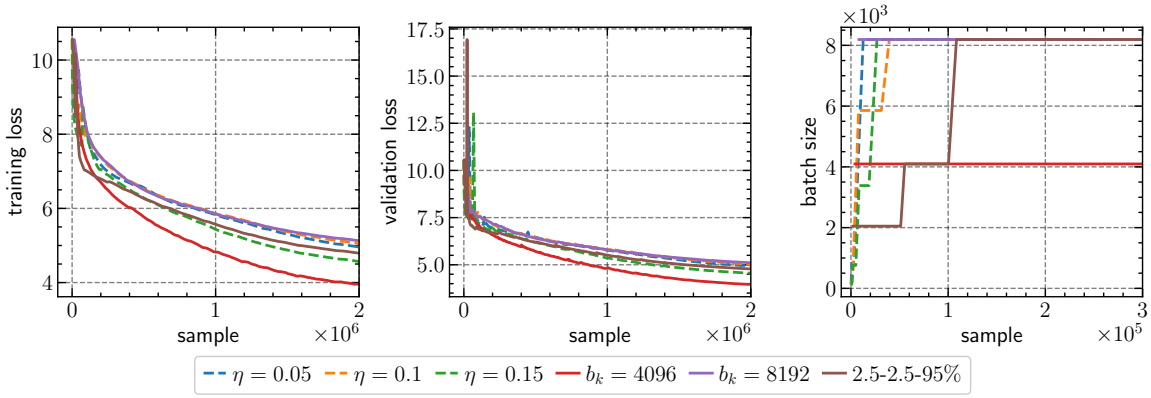

Figure 4: Training loss, validation loss and batch size schedule for OpenLlama 3B

5.4. Further Discussions of Experimental Results

The effect of η. The hyperparameter η in the adaptive batch size schedules has the effect of controlling the probability of *obtaining a descent direction* and hence *increasing the batch size*. Obviously, choosing a right value of η is vital for our method to succeed. Across all three sets of experiments of different model scales, we found that larger values of η generally lead to more gradual batch size increments, but smaller values would allow full utilization of available compute resources at earlier stages of training but might defeat the prupose of adaptive batch sizes. Note that η also varies with the

base learning rate α and the quality of the training datasets. In the series of works of adaptive sampling methods [10, 21, 77], there are in-depth discussions on choosing the learning rate via some line-search procedures, which are however usually infeasible when training large deep neural networks.

Scaling law of critical batch size. We conjecture that there are more general scaling laws of the critical batch size (see e.g., [56, 78–80]) in relation to η which controls gradient approximation quality and the scale of gradient noise. For most choices of η in the three sets of experiments, we choose η small enough so that global batch sizes increase rapidly and reach the maximum possible values. However, in Figure 2, when $\eta = 0.15$, the final batch size is around 3800, which might be the critical batch size at this value of η. It is thus crucial to understand the notion of critical batch sizes through the lens of gradient approximation quality and we leave this for future work.

6. Concluding Remarks

We create an efficient PyTorch `FSDP` implementation of the norm test for large-scale distributed training, focusing on hardware use and ease of development. Our implementation shows that adaptive batch size schedules can pretrain Llama 2 language models with up to 3 billion parameters using few GPUs [50]. Furthermore, we provide convergence guarantees of the norm test for ADAM, suggesting that our proposed adaptive batch size schedules are not only practically feasible, but also theoretically principled. Due to its generality, versatility, and scalability, we foresee extensive use of the adaptive batch size schedules in pretraining large transformer models like vision transformers (VIT) [81] and autoregressive image models (AIM) [82]. We emphasize our attention on a PyTorch `FSDP` approach due to its integration with PyTorch. However, a more advanced implementation of the adaptive batch size schedules, using a new version of PyTorch `FSDP` (`FSDP2`) and tensor parallelism via PyTorch `DTensor` (Distributed Tensor), as well as availing of stronger computational hardware, will significantly enhance the scalability of the method for training models exceeding 7B parameters with 2D, 3D or even 4D parallelism. For further exploration, we refer readers to the `torchtitan` [83] and `lingua` [84] repositories. Furthermore, while our current implementation is based on PyTorch `FSDP`, but is readily extendable to other deep learning frameworks such as JAX [19] with `FSDP` and/or GShard [85].

Limitations. In this work, we are primarily concerned with model generalization performance measured by validation loss without any evaluation on downstream benchmarks. The main reason for this is that we did not fully pretrain the models for sufficient number of tokens, implying that these models will not be competitive on downstream benchmarks. However, we expect that models fully pretrained with our proposed schedules will achieve very competitive performance on the evaluation of downstream benchmarks. We also remark that we can also incorporate other paradigms of parallelism such as pipeline and context parallelism with our proposed scheme, leading to 4D parallelism (data, tensor, pipeline, context parallel) for large-scale pretraining. While not supported in our current implementation, this can be achieved using the recent library `picotron` [86] or the more sophisticated library Megatron-LM [22]. We leave this implementation for future work.

Acknowledgments

Tim Tsz-Kit Lau would like to thank Zhihan Zhou for helpful discussion on the experimental setup. The research of Han Liu is supported by NIH R01LM01372201, NSF RI 1840857, NSF TRIPOD 1740735, NSF DMS1454377-CAREER, NSF IIS 1546482, along with an Alfred P. Sloan Fellowship. The research of Mladen Kolar is supported in part by NSF ECCS-2216912. This research is supported in part through the computational resources and staff contributions provided for the Quest high performance computing facility at Northwestern University which is jointly supported by the Office of the Provost, the Office for Research, and Northwestern University Information Technology. This research is also supported in part through the computational resources and staff contributions provided for the Data Science Institute at the University of Chicago, through its AI + Science Research Funding Initiatives.

References

[1] Llama Team, AI @ Meta. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[2] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer Berlin Heidelberg, 2002.

[3] 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. In *International Conference on Learning Representations* (*ICLR*), 2017.

[4] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2017.

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

[6] Samuel L. Smith and Quoc V. Le. A Bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations* (*ICLR*), 2018.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2017.

[8] Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Joshua M. Susskind. Stabilizing transformer training by preventing attention entropy collapse. In *Proceedings of the International Conference on Machine Learning* (*ICML*), 2023.

[9] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *International Conference on Learning Representations* (*ICLR*), 2024.

[10] Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical Programming*, 134(1):127–155, 2012.

[11] Michael P. Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

[12] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Big batch SGD: Automated inference using adaptive batch sizes. *arXiv preprint arXiv:1610.05792*, 2016.

[13] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Automated inference with adaptive batches. In *Proceedings of the International Conference on Artificial Intelligence and Statistics* (*AISTATS*), 2017.

[14] Tim Tsz-Kit Lau, Han Liu, and Mladen Kolar. AdAdaGrad: Adaptive batch size schemes for adaptive gradient methods. *arXiv preprint arXiv:2402.11215*, 2024.

[15] Tim Tsz-Kit Lau, Weijian Li, Chenwei Xu, Han Liu, and Mladen Kolar. Communication-efficient adaptive batch size strategies for distributed local gradient methods. *arXiv preprint arXiv:2406.13936*, 2024.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2012.

[17] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. PyTorch distributed: Experiences on accelerating data parallel training. In *Proceedings of the VLDB Endowment*, 2020.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

[19] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[20] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of Machine Learning and Systems* (*MLSys*), 2018.

[21] Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, 2018.

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[23] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. In *Proceedings of the VLDB Endowment*, 2023.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2019.

[25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[26] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

[27] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In *Proceedings of Machine Learning and Systems* (*MLSys*), 2023.

[28] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron

to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[29] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[30] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[31] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[32] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious. *arXiv preprint arXiv:2501.00656*, 2025.

[33] The Mosaic Research Team. Introducing DBRX: A new state-of-the-art open LLM. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024.

[34] Yan Pan and Yuanzhi Li. Toward understanding why Adam converges faster than SGD for transformers. *arXiv preprint arXiv:2306.00204*, 2023.

[35] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between SGD and Adam on transformers, but sign descent might be. In *International Conference on Learning Representations* (*ICLR*), 2023.

[36] Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why Adam outperforms gradient descent on language models. *arXiv preprint arXiv:2402.19449*, 2024.

[37] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need Adam: A Hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024.

[38] 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.

[39] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[40] 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. In *International Conference on Learning Representations* (*ICLR*), 2020.

[41] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[42] Raul Puri, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro. Large scale language modeling: Converging on 40GB of text in four hours. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018.

[43] BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ajibade, Bharat Saxena,

Carlos Muñoz Ferrandis, Daniel McDuff, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[44] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[45] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, JunTao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

[46] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[47] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin,

Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[48] Google DeepMind Gemma Team. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[49] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[50] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen

Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[51] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. DeepSeek-V2: A strong, economical, and efficient Mixture-of-Experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[52] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu

Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[53] Marco Bellagente, Jonathan Tow, Dakota Mahan, Duy Phung, Maksym Zhuravinskyi, Reshinth Adithyan, James Baicoianu, Ben Brooks, Nathan Cooper, Ashish Datta, Meng Lee, Emad Mostaque, Michael Pieler, Nikhil Pinnaparju, Paulo Rocha, Harry Saini, Hannah Teufel, Niccolo Zanichelli, and Carlos Riquelme. Stable LM 2 1.6B technical report. *arXiv preprint arXiv:2402.17834*, 2024.

[54] 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.

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

[56] 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.

[57] Petr Ostroukhov, Aigerim Zhumabayeva, Chulu Xiang, Alexander Gasnikov, Martin Takáč, and Dmitry Kamzolov. AdaBatchGrad: Combining adaptive batch size and adaptive step size. *arXiv preprint arXiv:2402.05264*, 2024.

[58] Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (*UAI*), 2017.

[59] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

[60] Heyang Qin, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. SimiGrad: Fine-grained adaptive batching for large scale training using gradient similarity measurement. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2021.

[61] Tyler Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. AdaScale SGD: A user-friendly algorithm for distributed training. In *Proceedings of the International Conference on Machine Learning* (*ICML*), 2020.

[62] Jacob Hilton, Jie Tang, and John Schulman. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*, 2023.

[63] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2020.

[64] Jupinder Parmar, Shrimai Prabhumoye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, Vibhu Jawa, Jiwei Liu, Ameya Mahabaleshwarkar, Osvald Nitski, Annika Brundyn, James Maki, Miguel Martinez, Jiaxuan You, John Kamalu, Patrick LeGresley, Denys Fridman, Jared Casper, Ashwath Aithal, Oleksii Kuchaiev, Mohammad Shoeybi, Jonathan Cohen, and Bryan Catanzaro. Nemotron-4 15B technical report. *arXiv preprint arXiv:2402.16819*, 2024.

[65] Conglong Li, Minjia Zhang, and Yuxiong He. The stability-efficiency dilemma: Investigating sequence length warmup for training GPT models. *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022.

[66] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Chia-Yuan Chang, and Xia Hu. GrowLength: Accelerating LLMs pretraining by progressively growing training length. *arXiv preprint arXiv:2310.00576*, 2023.

[67] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[68] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing billion-scale model training. In *USENIX Annual Technical Conference* (*USENIX ATC*), 2021.

[69] Diederik P. Kingma and Jimmy Lei Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations* (*ICLR*), 2015.

[70] Bohan Wang, Jingwen Fu, Huishuai Zhang, Nanning Zheng, and Wei Chen. Closing the gap between the upper bound and lower bound of Adam's iteration complexity. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023.

[71] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations* (*ICLR*), 2019.

[72] Ken Wang. MicroLlama-300M. https://github.com/keeeeenw/MicroLlama, 2024.

[73] Xinyang Geng and Hao Liu. OpenLLaMA: An open reproduction of LLaMA, May 2023. URL https://github.com/openlm-research/open_llama.

[74] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[75] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. URL https://github.com/Lightning-AI/lightning. Version 2.0.8.

[76] Lightning AI. LitGPT, 2023. URL https://github.com/Lightning-AI/litgpt.

[77] Raghu Bollapragada and Stefan M. Wild. Adaptive sampling quasi-Newton methods for zeroth-order stochastic optimization. *Mathematical Programming Computation*, 15(2):327–364, 2023.

[78] Hui Su, Zhi Tian, Xiaoyu Shen, and Xunliang Cai. Unraveling the mystery of scaling laws: Part I. *arXiv preprint arXiv:2403.06563*, 2024.

[79] Antonio Sclocchi and Matthieu Wyart. On the different regimes of stochastic gradient descent. *Proceedings of the National Academy of Sciences*, 121(9):e2316301121, 2024.

[80] 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? In *International Conference on Learning Representations* (*ICLR*), 2025.

[81] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations* (*ICLR*), 2021.

[82] Alaaeldin El-Nouby, Michal Klein, Shuangfei Zhai, Miguel Angel Bautista, Alexander Toshev, Vaishaal Shankar, Joshua M. Susskind, and Armand Joulin. Scalable pre-training of large autoregressive image models. In *Proceedings of the International Conference on Machine Learning* (*ICML*), 2024.

[83] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. TorchTitan: One-stop PyTorch native solution for production ready LLM pre-training. In *International Conference on Learning Representations (ICLR)*, 2025.

[84] Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

[85] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021.

[86] Haojun Zhao and Ferdinand Mom. Picotron: Distributed training framework for education and research experimentation, 2025. URL https://github.com/huggingface/picotron.

[87] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[88] Ahmed Khaled and Peter Richtárik. Better theory for SGD in the nonconvex world. *Transactions on Machine Learning Research*, 2023.

Appendix

Contents

A. Additional Details of The Proposed Algorithm

Note that the use of PyTorch `FSDP` does not lead to significance difference in the implementation of the norm test compared to its `DDP` implementation. We assume that the gradients of different parameter shards are concatenated together in the following computation to simplify the representation.

A.1. The Overall Algorithm

Algorithm 1 DDP-Norm or FSDP-Norm for AdamW

Input: $w_1 \in \mathbb{R}^d$, $m_0 = v_0 = \mathbf{0}_d \in \mathbb{R}^d$, $(\alpha, \lambda, \varepsilon, \beta_1, \beta_2) \in (0, \infty)^5$, $\mathcal{D}_n = \{z_i\}_{i \in [\![n]\!]} \subset \mathcal{Z}$, number of workers $J \in \mathbb{N}^*$, number of gradient accumulation steps $M \in \mathbb{N}^*$, number of training samples $N \in \mathbb{N}^*$, step counter $k = 1$, processed sample counter $i = 0$, initial (global) batch size b_0, initial microbatch size $b_{0,J}^M = b_0/(JM)$

while $i < N$ **do**
- Sample the i.i.d. data batch (indices) $\mathcal{B}_k$ uniformly from $[\![n]\!]$ of size $b_k \coloneqq |\mathcal{B}_k|$
- Split $\mathcal{B}_k$ evenly to each worker $j \in [\![J]\!]$, each with $\mathcal{B}_{k,j}$ of size $b_{k,J}$
- **for all** $j = 1, \ldots, J$ in parallel **do**
  - Split $\mathcal{B}_{k,j}$ evenly to each gradient accumulation step $m \in [\![M]\!]$, each with $\mathcal{B}_{k,j}^m$ of size $b_{k,J}^M$
  - Initialize $\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k) = \mathbf{0}_d$
  - **for** $m = 1, \ldots, M$ **do**
    - Compute $\frac{1}{M} \nabla \mathscr{L}_{\mathcal{B}_{k,j}^m}(w_k)$
    - Accumulate gradients $\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k) \leftarrow \nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k) + \frac{1}{M} \nabla \mathscr{L}_{\mathcal{B}_{k,j}^m}(w_k)$
  - **end for**
- **end for**
- Compute the batch gradient $g_k \coloneqq \nabla \mathscr{L}_{\mathcal{B}_k}(w_k)$ with *all-reduce*
- Compute the approximate gradient variance $\widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k))$ with *all-reduce*

$$\widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k)) \coloneqq \frac{1}{J} \sum_{j \in [\![J]\!]} \left(\nabla \mathscr{L}_{\mathcal{B}_{k,j}}(w_k) - g_k \right)^2$$

- Compute the approximate norm test statistic

$$\mathsf{T}_k \equiv \mathsf{T}(w_k; \mathcal{B}_k, \eta) \coloneqq \frac{\left\| \widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k)) \right\|_1}{\eta^2 \|g_k\|^2}$$

- **if** $\mathsf{T}_k > b_k$ **then**
  - Increase the next global batch size $b_{k+1} = \lceil \mathsf{T}_k \rceil$
  - Round up the microbatch size $b_{k+1,J}^M = \lceil b_{k+1}/(JM) \rceil$
  - Update the minibatch size $b_{k+1,J} = M b_{k+1,J}^M$
  - Update the global batch size again $b_{k+1} = J b_{k+1,J}$
- **else**
  - $b_{k+1} = b_k$
- **end if**
- $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$ ▷ AdamW
- $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
- $\widehat{m}_k = m_k \odot (1 - \beta_1^k)^{-1}$
- $\widehat{v}_k = v_k \odot (1 - \beta_2^k)^{-1}$
- $w_{k+1} = (1 - \alpha\lambda) w_k - \alpha \widehat{m}_k \odot (\widehat{v}_k^{1/2} + \varepsilon)^{-1}$
- $k \leftarrow k + 1$
- $i \leftarrow i + b_k$

end while

B. Proofs of Main Text

We give a brief sketch of the omitted proof of the main text in this section. Notice that the convergence analysis of the norm test for ADAM largely follows that in [70], where more details and remarks of the analysis and rationales of its derivation can be found.

Remark B.1. Despite the similarity of the proof techniques, we emphasize that our setting requires less restrictive assumptions. While Wang et al. [70] assume the stochastic oracle of the gradient (i.e., batch gradient in our case) has coordinate-wise affine variance, i.e., for any batch of samples $\mathcal{B} \subseteq \mathcal{D}_n$ and $(\sigma, \tau) \in (0, \infty)^2$, we have

$$(\forall i \in [\![d]\!])(\forall w \in \mathbb{R}^d) \quad \mathbb{E}\big[(\partial_i \mathscr{L}_{\mathcal{B}}(w))^2\big] \leqslant \sigma^2 + \tau^2 (\partial_i \mathscr{L}(w))^2.$$

We do not impose this global condition which is often difficult to verify in practical scenarios, but instead we increase the (next) batch size such that the condition of the coordinate-wise (exact variance) norm test with constant $\eta \in (0, 1)$ is satisfied at the current iterate $w_k \in \mathbb{R}^d$ with the current batch $\mathcal{B}_k \subseteq \mathcal{D}_n$:

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\big[(\partial_i \mathscr{L}_{\mathcal{B}_k}(w_k) - \partial_i \mathscr{L}(w_k))^2\big] \leqslant \eta^2 (\partial_i \mathscr{L}(w_k))^2,$$

which implies

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\big[(\partial_i \mathscr{L}_{\mathcal{B}_k}(w_k))^2\big] \leqslant (1 + \eta^2)(\partial_i \mathscr{L}(w_k))^2,$$

which is also known as the coordinate-wise *expected strong growth* (E-SG) condition [14]. Note that the coordinate-wise (E-SG) condition implies the coordinate-wise *relaxed growth* (RG) condition [87], adopting the nomenclature in [88]:

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\big[(\partial_i \mathscr{L}_{\mathcal{B}_k}(w_k))^2\big] \leqslant \sigma^2 + \tau^2 (\partial_i \mathscr{L}(w_k))^2,$$

where $\tau^2 = 1 + \eta^2$ and $\sigma \in (0, \infty)$. Recall that we only require such a condition to hold at the current iterate w_k with the current batch $\mathcal{B}_k$, through the enforcement of the coordinate-wise (exact variance) norm test. Even though the exact variance test is not implemented in practice but its approximate version instead, this is often a good heuristic to justify the convergence of the test.

Additional notation. To simplify notation, we denote the full gradient by $\mathfrak{G}_k \coloneqq \nabla \mathscr{L}(w_k)$, and its ith coordinate by $\mathfrak{G}_{k,i} \coloneqq \partial_i \mathscr{L}(w_k)$.

B.1. Technical Lemmas

We state without proof the following technical lemmas from [70].

Lemma B.1. *Let $0 < \beta_1^2 < \beta_2 < 1$ and consider a sequence of real numbers $(a_n)_{n \in \mathbb{N}^*} \subset \mathbb{R}$. Let $b_0 > 0$, $b_k = \beta_2 b_{k-1} + (1 - \beta_2) a_k^2$, $c_0 = 0$ and $c_k = \beta_1 c_{k-1} + (1 - \beta_1) a_k$. We have the following inequality*

$$\sum_{k=1}^{K} \frac{|c_k|^2}{b_k} \leqslant \frac{(1 - \beta_1)^2}{(1 - \beta_2)(1 - \beta_1/\sqrt{\beta_2})^2} \left(\log\left(\frac{b_K}{b_0} \right) - K \log \beta_2 \right). \tag{B.1}$$

Lemma B.2. *Consider the ADAM iterates $(w_k)_{k \in \mathbb{N}^*}$ generated by (6). Then we have*

$$(\forall k \in \mathbb{N}^*) \quad |w_{k+1,i} - w_{k,i}| \leqslant \alpha \frac{1 - \beta_1}{\sqrt{1 - \beta_2}\sqrt{1 - \beta_1^2/\beta_2}} \leqslant \alpha \frac{1 - \beta_1}{\sqrt{1 - \beta_2}\sqrt{1 - \beta_1/\beta_2}}.$$

Proof Sketch. This is due to the definition of the ADAM iterate and Cauchy–Schwarz's inequality. □

B.2. Proof of Theorem 1

Since the proof of Theorem 1 is highly similar to that in [70], we just provide a proof sketch. We state the formal theorem as follows.

Theorem B.1 (Formal version of Theorem 1)**.** *Suppose that Assumption 1 holds. Let $(w_k)_{k\in\mathbb{N}^*}$ be the* Adam *iterates generated by (6), where the batch size $b_k := |\mathcal{B}_k|$ is chosen such that the coordinate-wise (exact variance) norm test with constant $\eta \in (0,1)$ is satisfied at each iteration $k \in \mathbb{N}^*$. Then, if $0 < \beta_1 \leqslant \sqrt{\beta_2} - 8(1+\eta^2)(1-\beta_2)/\beta_2^2$ and $\beta_2 \in (0,1)$, we have*

$$\begin{aligned}
&\sum_{k=1}^{K} \mathbb{E}[\|\nabla \mathscr{L}(w_k)\|] \\
&\qquad \leqslant \sqrt{c_2 + 2c_1 \sum_{i=1}^{d} \left[\log\left(2(K+1) \sum_{i=1}^{d} \sqrt{v_{0,i} + \sigma^2} + 24d \frac{\tau^2 c_1}{\sqrt{\beta_2}} \log\left(d \frac{\tau^2 c_1}{\sqrt{\beta_2}} \right) + \frac{12\tau^2}{\sqrt{\beta_2}} c_2 \right) \right]} \\
&\qquad\qquad \times \sqrt{2(K+1) \sum_{i=1}^{d} \sqrt{v_{0,i} + \sigma^2} + 24d \frac{\tau^2 c_1}{\sqrt{\beta_2}} \log\left(d \frac{\tau^2 c_1}{\sqrt{\beta_2}} \right) + \frac{12\tau^2}{\sqrt{\beta_2}} c_2}, \qquad \text{(B.2)}
\end{aligned}$$

where $v_{0,i}$ is the i-coordinate of v_0, $\tau^2 = 1 + \eta^2$, $\sigma \in (0, \infty)$,

$$\begin{aligned}
c_1 &:= \frac{32L\alpha\left(1 + \beta_1/\sqrt{\beta_2}\right)^3}{(1-\beta_2)\left(1 - \beta_1/\sqrt{\beta_2}\right)^3} + \frac{16\beta_1^2 \sigma (1-\beta_1)}{\beta_2 \sqrt{1-\beta_2}\left(1 - \beta_1/\sqrt{\beta_2}\right)^3} + \frac{64(1+\sigma^2)\sigma^2 L^2 \alpha^2 d}{\beta_2^2 \left(1 - \beta_1/\sqrt{\beta_2}\right)^4 \sigma (1-\beta_2)^{3/2}}, \\
c_2 &:= \frac{8(1 - \beta_1/\sqrt{\beta_2})}{\alpha(1-\beta_1)} + \frac{32}{\beta_2 \left(1 - \beta_1/\sqrt{\beta_2}\right)^2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathfrak{G}_{1,i}^2}{\sqrt{\widetilde{v}_{1,i}}} \right] + 2c_1 \sum_{i=1}^{d} \left(\log\left(\frac{1}{\sqrt{\beta_2 v_{0,i}}} \right) - K \log \beta_2 \right), \\
u_k &:= \frac{w_k - \beta_1 w_{k-1}/\sqrt{\beta_2}}{1 - \beta_1/\sqrt{\beta_2}}.
\end{aligned}$$

The proof consists of deriving a descent lemma on the sequence $u_k := \frac{w_k - \beta_1 w_{k-1}/\sqrt{\beta_2}}{1-\beta_1/\sqrt{\beta_2}}$.

Lemma B.3. *Suppose that all the assumptions in Theorem B.1 hold. We also define the function $\varphi_k := \mathbb{E}\left[-\alpha \left\langle \mathfrak{G}_k, \mathfrak{G}_k \odot \widetilde{v}_{k+1}^{-1/2} \right\rangle \right]$. Then we have*

$$\begin{aligned}
&\mathbb{E}[\mathscr{L}(u_{k+1})] \\
&\leqslant \mathbb{E}[\mathscr{L}(u_k)] - \frac{\alpha(1-\beta_1)}{4(1-\beta_1/\sqrt{\beta_2})} \mathbb{E}\left[-\alpha \left\langle \mathfrak{G}_k, \mathfrak{G}_k \odot \widetilde{v}_k^{-1/2} \right\rangle \right] + \frac{2\alpha\sigma\sqrt{1-\beta_2}}{(1-\beta_1^2/\beta_2)^2} \sum_{i=1}^{d} \left[\frac{g_{k,i}^2}{v_{k,i}} \right] \\
&\quad + \frac{4\alpha\tau^2}{(1-\beta_1/\sqrt{\beta_2})^2 \sqrt{\beta_2}} \sum_{i=1}^{d} \mathbb{E}\left[\frac{1}{\beta_2} \varphi_{k-1} - \varphi_k \right] + \sum_{i=1}^{d} \frac{2\alpha\sigma\sqrt{1-\beta_2}}{(1-\beta_1)(1-\beta_1/\sqrt{\beta_2})} \mathbb{E}\left[\frac{m_{k,i}^2}{v_{k,i}} \right] \\
&\quad + \frac{64d(1+\tau^2)\tau^2 L^2 \alpha^3}{\beta_2^2 (1-\beta_1/\sqrt{\beta_2})^3 (1-\beta_1)\sigma\sqrt{1-\beta_2}} \cdot \mathbb{E}\left[\left\| m_{k-1} \odot v_{k-1}^{-1/2} \right\|^2 \right] \\
&\quad + \sum_{i=1}^{d} \frac{2\alpha\sigma\beta_1^2\sqrt{1-\beta_2}}{\beta_2(1-\beta_1)(1-\beta_1/\sqrt{\beta_2})} \mathbb{E}\left[\frac{m_{k-1,i}^2}{v_{k-1,i}} \right] \\
&\quad + L\mathbb{E}\left[4\alpha^2 \left(\frac{\beta_1/\sqrt{\beta_2}}{1-\beta_1/\sqrt{\beta_2}} \right)^2 \left\| m_{k-1} \odot v_{k-1}^{-1/2} \right\|^2 + 3\alpha^2 \left(\frac{1}{1-\beta_1/\sqrt{\beta_2}} \right)^2 \left\| m_k \odot v_k^{-1/2} \right\|^2 \right].
\end{aligned}$$

Proof Sketch. This bound is derived by bounding the "first-order term" and the "second-order term", similar to the derivation of a descent lemma for Lipschitz smooth functions but on the sequence $(u_k)_{k\in\mathbb{N}^*}$. □

Lemma B.4. *Suppose that all the assumptions in Theorem B.1 hold. Then we have*

$$\sum_{k=1}^{K+1} \sum_{i=1}^{d} \mathbb{E}[\widetilde{v}_{k,i}^{1/2}] \leqslant 2(K+1) \sum_{i=1}^{d} \sqrt{v_{0,i} + \sigma^2} + \frac{24d\tau^2 c_1}{\sqrt{\beta_2}} \log\left(\frac{d\tau^2 c_1}{\sqrt{\beta_2}} \right) + \frac{12\tau^2 c_2}{\sqrt{\beta_2}}.$$

Proof Sketch. This bound is derived by a *divide-and-conquer* approach, considering the cases $|\mathfrak{G}_{k,i}| \geqslant \sigma/\tau$ and $|\mathfrak{G}_{k,i}| \leqslant \sigma/\tau$ respectively. □

Proof Sketch of Theorem 1. The final bound is derived by first summing the inequality in Lemma B.3 with the assumed condition of (β_1, β_2). Further application of Lemma B.1, Cauchy–Schwarz's inequality and Lemma B.4 implies the desired result. □

C. Details of Numerical Experiments

We provide a summary table for the architecture of the language models we pretrained. More details of these models can be found in [30, 72, 73].

Model	MicroLlama 300M	TinyLlama 1.1B	OpenLlama 3B
n_{params}	304.6M	1.1B	3.4B
d_{model}	2048	2048	2048
n_{layers}	12	22	26
n_{heads}	12	32	32
d_{head}	64	64	100

Table 4: Specifications of models

We also summarize the training hyperparameters of the three sets of experiments in the following tables.

C.1. MicroLlama 300M

Model	MicroLlama 300M
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	2048
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	256
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table 5: Training hyperparameters for MicroLlama 300M

C.2. TinyLlama 1.1B

Model	TinyLlama 1.1B
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	2048
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	128
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table 6: Training hyperparameters for TinyLlama 1.1B

C.3. OpenLlama 3B

Model	OpenLlama 3B
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	512
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	128
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table 7: Training hyperparameters for OpenLlama 3B