arXiv:2505.23725v3 [cs.LG] 2 Jun 2026

MuLoCo: Muon is a Practical Inner Optimizer for DiLoCo

Benjamin Thérien[1,2,3,*], **Xiaolong Huang**[2,4], **Aaron Defazio**[1], **Irina Rish**[2,3], **Eugene Belilovsky**[2,4]

[1]FAIR at Meta, [2]Mila, [3]Université de Montréal, [4]Concordia University
[*]Work done at Meta

DiLoCo is a powerful framework for training large language models (LLMs), enabling larger optimal batch sizes and increased accelerator utilization under networking constraints. However, DiLoCo's performance has been shown to degrade as the number of workers (K) increases (Charles et al., 2025). In this work, we posit that a related but often overlooked factor in DiLoCo's behavior is the choice of inner optimizer, which shapes the pseudogradient used by the outer optimizer. Given the recent success of Muon relative to AdamW for data parallel (DP) training, we examine how Muon's normalized optimizer steps can affect the pseudogradient's quality. We find that, relative to AdamW, Muon yields more *directionally correct* pseudogradients as the number of workers (K) increases. In our experiments pre-training language models, we conduct extensive hyperparameter tuning across 150M, 416M, 914M, 1.76B, and 3.1B models for DiLoCo, MuLoCo, AdamW DP, and Muon DP. Consistently across all scales, we find that with $K \geq 1$ workers, MuLoCo (Muon inner optimizer DiLoCo) achieves superior performance to DiLoCo in absolute terms and for $K > 2$ it outperforms DiLoCo relative to their data parallel baselines, while being compatible with quantization, streaming, and long synchronization intervals. At $K = 1$, we find that MuLoCo can even outperform the data-parallel gold standard while having larger critical batch sizes. Finally, we extrapolate optimal hyperparameters to 15B scale and train a model with each method (six in total) using $K = 1$ and $K = 16$ workers. We find that $K = 16$ MuLoCo nearly matches single-worker performance at this scale, while MuLoCo $K = 1$ matches the best performing baseline while using a much larger 16M token batch size.

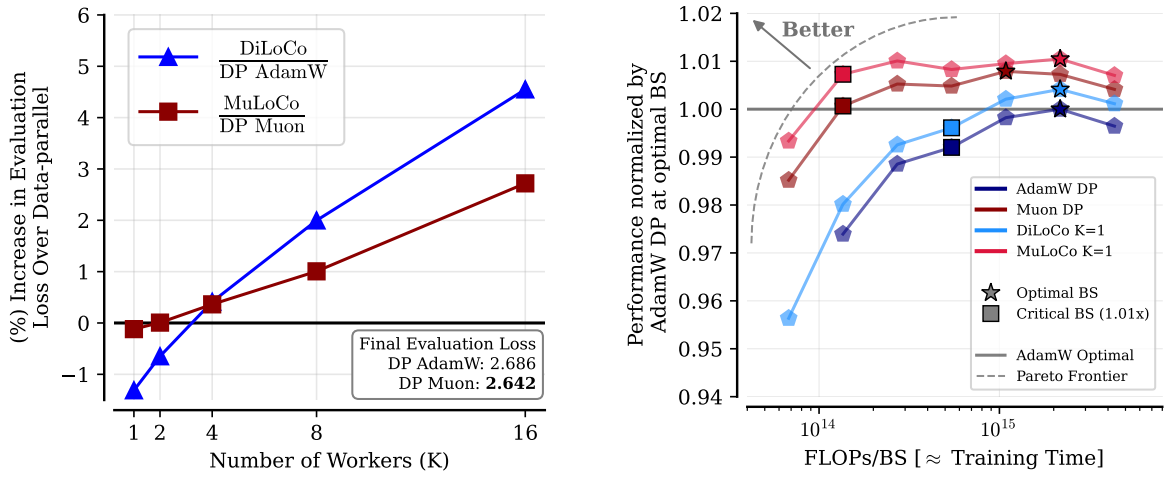
Correspondence: Benjamin Thérien at benjamin.therien@umontreal.ca
Code: https://github.com/facebookresearch/MuLoCo
Website: https://bentherien.github.io/muloco-1

Meta

1 Introduction

In Large Language Model (LLM) pre-training, it is now well established that scaling training data and model parameters in tandem at a ratio of roughly 20 tokens per parameter (TPP) is compute optimal (Hoffmann et al., 2022; Hägele et al., 2024; Besiroglu et al., 2024; Dey et al., 2023; Schaeffer et al., 2025). With the largest existing training runs reaching $10 - 40$T tokens (Yang et al., 2025; Team et al., 2025; DeepSeek-AI, 2024) and estimates of available data reaching 300T (Villalobos et al., 2024), it is reasonable to assume that the largest compute optimal models trained in the near future will have 2-15T parameters. Efficiently training at this scale is bottlenecked by two requirements of data-parallel training: (a) the need to take a large number of gradient descent steps sequentially, leading to long wall-clock training times, and (b) the need to communicate among all model replicas at each optimization step, leading to decreased compute utilization across tens of thousands of accelerators. A straightforward remedy to (a) is training at large batch sizes. However, at the largest scales, this is likely to exacerbate (b) since there is a limit to the number of accelerators that are tightly interconnected in typical deployments (NVIDIA, 2024).

The motivation for training at large batch sizes is clear: leveraging more accelerators in parallel leads to faster wall-clock training times for the same amount of data. At the largest model scales, computing gradients on a single training example can be expensive. Without large batch training across enough accelerators in parallel, practitioners are required to (1) accumulate gradients, leading to longer step times, or (2) take more gradient descent steps sequentially. The critical batch size (CBS) (McCandlish et al., 2018; Merrill et al., 2025; Zhang et al., 2025; Bergsma et al., 2025) is known as the largest batch size that can be used to achieve performance



(a) MuLoCo outperforms DiLoCo in absolute terms and continues to do so at larger worker counts even after normalizing by their respective data-parallel baselines.

(b) MuLoCo K=1 acheives a Pareto-Optimal Performance-Time Tradeoff. All datapoints are FLOP-matched.

Figure 1 Subfigure (a) reports the percentage increase in loss for DiLoCo and MuLoCo relative to their DP baselines. We observe that the relative performance of each method degrades as K is increased, but that MuLoCo shows clear improvements over DiLoCo relative to their data parallel baselines as workers are increased. **Subfigure (b)** reports the performance of different optimizers in an iso-FLOP setting as a function of FLOPs/Batch Size (a proxy for training time). The optimal batch size (B_{opt}) is the batch size achieving the best performance, while the critical batch size (B_{crit}) is taken to be the largest batch size achieving a final evaluation loss that is: $L(B_{\text{crit}}) \leq 1.01 \times L(B_{\text{opt}})$. We observe that at the 3.1B scale, our proposed MuLoCo K=1 achieves stronger performance than other optimizers while also being compatible with much larger batch sizes.

nearly as strong as the optimal (best performing) batch size. Therefore, training at the CBS with enough devices in parallel can lead to linear speedups in training time with nearly no performance degradation, as long as communication costs do not grow with the number of devices. However, when training at the largest model scales or across poorly interconnected accelerators, this is rarely the case.

Due to the large communication costs associated with data-parallel (DP) training at large model scales or under bandwidth constraints, a number of communication-efficient training algorithms have been proposed. Most recently, DiLoCo (Douillard et al., 2023) has gained attention due to its competitive performance with DP pre-training of LLMs and its larger optimal batch sizes (Charles et al., 2025). DiLoCo independently trains a single model with DP AdamW across K workers in parallel while only communicating across workers every H gradient descent steps. When individual workers span highly interconnected compute pools (e.g., single nodes, racks, adjacent racks, or entire datacenters for the largest models) with significantly faster bandwidth within pools than across them, DiLoCo can lead to significant speedups relative to DP training across the same number of accelerators. However, Charles et al. (2025) shows that the performance of DiLoCo degrades as the number of workers is increased, leading to questions of its utility at large worker counts.

In this work, we seek to address DiLoCo's worker scaling problem while simultaneously increasing its critical batch size. Our core hypothesis is that this degradation is related to the choice of inner optimizer, which shapes the pseudogradient used by the outer optimizer. Given the recent success of Muon relative to AdamW for LLM pre-training, its larger critical batch sizes, and its orthonormalized optimizers steps that are substantially different from those of AdamW (Jordan et al., 2024; Liu et al., 2025; Shah et al., 2025a), we conjecture that, beyond its improved convergence, Muon may offer additional advantages in terms of compressibility, parallelization, and performance within a Local SGD framework. Given the potential of Muon to be a powerful inner optimizer for DiLoCo, in what follows, we systematically compare MuLoCo (Muon inner optimizer) to DiLoCo in settings using a large number of workers (K), long communication intervals (H), when using compressed communication, and when using streaming updates. We further scale compute following a chinchilla optimal prescription and examine whether our most interesting findings hold at up to 15B parameter

scale. Our contributions can be summarized as follows:

MuLoCo $K > 1$

- We establish that MuLoCo's performance scales better with workers than DiLoCo. Specifically, beyond outperforming DiLoCo in absolute terms, MuLoCo outperforms DiLoCo at $K > 2$ workers even when normalizing by the performance of their respective data parallel baselines. We find that this advantage is maintained at scale.
- We demonstrate that MuLoCo's performance at large worker counts (K=16) improves relative to data-parallel baselines as the training scale increases. Indeed, at 15B scale, K=16 MuLoCo matches the final performance of DP AdamW on language model evaluation benchmarks.
- We demonstrate that MuLoCo outperforms DiLoCo under long synchronization intervals, under pseudo-gradient compression, with partitioned communication, and in wall-clock training time, while having lower memory complexity.
- At 3.1B scale, we find that MuLoCo's critical batch size is 2–4× larger than DiLoCo's, further reducing communication while increasing usable data parallelism.

MuLoCo $K = 1$

- At 3.1B scale, we find that K=1 MuLoCo outperforms DP Muon, DP AdamW, and K=1 DiLoCo, while having stronger performance and larger critical batch sizes—placing it on the Pareto frontier of performance vs. training time among the optimizers in our study.
- Our scaling laws predict that K=1 MuLoCo will retain its performance advantage and larger critical batch size over DP Muon at scale, leading to faster training at the largest model sizes. Our experiments at 15B validate this, showing that K=1 MuLoCo matches the best performing baselines while using a (4-16X) larger 16M token batch size.

Understanding MuLoCo's improved worker scaling

- Empirically, we find that Muon's orthonormalized steps lead to a more directionally aligned pseudogradient than AdamW and that the spectrum of AdamW's individual weight differences collapses during averaging.
- Theoretically, we link these findings, showing that the pseudogradient's nuclear norm is governed by its alignment with individual optimizer steps and the magnitude (Frobenius norm) of these steps. Our analysis reveals Muon's orthonormalized steps stabilize these magnitudes relative to AdamW.
- Our findings demonstrate that the structure of the inner-optimizer's step can have a non-trivial effect on pseudogradient quality.

2 Background

We briefly present the necessary background for understanding our empirical evaluation of MuLoCo. This includes implementation details of Muon, a review of DiLoCo, a review of partitioned communication, a review of common compression algorithms for neural networks, and a review of efficient communication collectives used in tandem with these compression algorithms.

Muon is a recently proposed (Jordan et al., 2024) second-order optimizer that takes orthonormalized steps. Specifically, at each step, Muon updates its momentum accumulator,

$$m_t = \beta m_{t-1} + g_t,$$

and applies five steps of *Newton–Schulz* iteration (Björck and Bowie, 1971; Kovarik, 1970) to produce an orthogonalized update. Specifically, the iteration refines

$$X_{t,j} = p(X_{t,j-1} X_{t,j-1}^\top) X_{t,j-1},$$

where $p(x) = ax + bx^3 + cx^5$ with (a, b, c) empirically tuned to be $3.4445, -4.7750, 2.0315$ (Jordan et al., 2024). This procedure effectively scales and rotates m_t toward the nearest orthogonal matrix, approximating $UV^\top$ when $m_t = U\Sigma V^\top$ is its singular value decomposition. The resulting orthogonalized update O_t is then rescaled by the learning rate and applied as $W_t = W_{t-1} - \eta O_t - \eta\lambda W_{t-1}$ with optional decoupled weight decay. In this work, we always use decoupled weight decay with Muon as it was shown to be important in maintaining Muon's convergence benefits at larger model sizes (Liu et al., 2025).

DiLoCo (Douillard et al., 2023) is a distributed optimization algorithm that independently trains a single model with DP AdamW across K workers in parallel while only communicating across workers every H gradient descent steps. When individual workers span highly interconnected compute pools (e.g., single nodes, racks, adjacent racks, or entire datacenters for the largest models) with significantly faster bandwidth within pools than across them, DiLoCo can lead to significant speedups relative to DP training across the same number of accelerators. We present the original DiLoCo algorithm and highlight our modifications to it in Algorithm 1 and 2.

Partitioned Communication in Streaming DiLoCo. Douillard et al. (2025) propose Streaming DiLoCo, a variant of DiLoCo which, among other modifications, reduces peak communication bandwidth by partitioning a model's layers into J subsets. With a communication interval H (J must divide H), each subset j is communicated at an assigned multiple $j \cdot H/J$ with $1 \leq j \leq J$ of the communication interval. In this work we evaluate the performance of MuLoCo under partitioned communication.

Compressed Communication. Compressing neural network gradients and parameters has been well studied in the context of reducing communication costs (Rothchild et al., 2020; Stich et al., 2018; Wang et al., 2023b; Peng et al., 2024). This work studies two common compression algorithms: quantization and sparsification. We now briefly discuss how these compressors are applied to a single weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$:

- **Top-k** sparsification involves keeping the $k\%$ largest magnitude entries of $\boldsymbol{W}$. The remaining entries are set to 0 and are not communicated. Note that one must still communicate the sparsity pattern (Wang et al., 2023b).
- **Quantization** involves mapping the entries of $\boldsymbol{W}$ to a much smaller representable range, such that they can be efficiently encoded using an offset and a limited number of discrete levels from a codebook. The number of bits determines the size of the codebook, while the codebook itself defines the quantized values used to approximate entries in $\boldsymbol{W}$. In *linear* quantization, these levels are uniformly spaced over a fixed range, whereas *statistical* (or non-uniform) quantization allocates levels based on the empirical distribution of $\boldsymbol{W}$, typically assigning higher resolution to more frequently occurring values.

Collectives for compressed communication. A ring all-reduce is bandwidth-optimal in terms of communication volume, achieving the minimum possible per-worker data transfer for dense reductions Wang et al. (2023a). However, exploiting compressed communication generally requires modifying the underlying collective. For example, Top-k sparsification necessitates an all-gather operation, whose bandwidth cost grows linearly with the number of workers.

For quantized communication, naively applying a ring-based reduce-scatter leads to error accumulation, since each hop requires a dequantize–reduce–quantize operation to preserve compression benefits. A natural remedy is to replace the ring reduce-scatter with an all-to-all reduce-scatter, which achieves the same aggregate bandwidth cost while performing reductions locally after receiving all peers' quantized pseudogradients (Wang et al., 2023a).

Unlike prior work (Douillard et al., 2025; Basu et al., 2019), which abstracts away the choice of communication primitive, we explicitly model an efficient all-to-all reduce-scatter followed by a ring all-gather in our experiments. This design requires a single dequantize–reduce–quantize operation between the two collectives, resulting in two quantization and two dequantization operations per pseudogradient communication. To the best of our knowledge, this regime has not been empirically evaluated before.

Error feedback (EF) is a powerful technique for mitigating the bias introduced by lossy compression (Karimireddy et al., 2019). EF works by persisting the difference between the original update and its compressed version and incorporating it into future communications. Concretely, given a compressor $\mathcal{C}(\cdot)$ and a residual accumulator $\mathcal{E}_i$, worker i updates its EF accumulator $\mathcal{E}_i \leftarrow \beta\,\mathcal{E}_i + \Delta_i$, communicates the compressed update $\tilde{\Delta}_i = \mathcal{C}(\mathcal{E}_i)$, and removes the communicated value $\mathcal{E}_i \leftarrow \mathcal{E}_i - \tilde{\Delta}_i$, thereby only persisting what was not communicated.

Error feedback has been shown to recover convergence guarantees comparable to uncompressed optimization for both sparsification and quantization under standard assumptions (Karimireddy et al., 2019). In our compression experiments, we ablate the effect of error feedback.

3 Related Work

We now contextualize our paper within the rich body of existing works on distributed and communication-efficient optimization (3.1,3.2), single-worker DiLoCo (3.3), and large batch training (3.4).

3.1 Distributed optimization with local steps

The idea of taking multiple local steps before communicating originated in the context of federated learning (McMahan et al., 2017), where the algorithm is known as Federated Averaging (FedAVG). In an i.i.d. setting, a variant of FedAVG known as Local SGD has been shown theoretically lead to communication savings (Stich, 2018). Models trained with LocalSGD were also found to have improved generalization relative to large batch training (Lin et al., 2018). Follow-up works have proposed more sophisticated server-side optimizers (Wang et al., 2019; Joseph et al., 2025; Reddi et al., 2021), while other research has focused on adaptive client-side optimizers (Zhou et al., 2024; Douillard et al., 2023; Sani et al., 2024). Notably, in the context of LLM pre-training, Douillard et al. (2023) propose DiLoCO, which uses SGD with Nesterov momentum as the server-side optimizer and AdamW on local workers. DiLoCo was demonstrated to scale to very large numbers of local steps ($H > 500$) (Douillard et al., 2023) and large models (Charles et al., 2025; Jaghouar et al., 2024b,a). Other recent works focus on studying DiLoCo variants that also communicate optimizer states, allowing them to prove convergence and improve performance at the cost of additional communication (Cheng and Glasgow, 2025; Iacob et al., 2026b,a; Jovanović et al., 2026). In a direct follow-up to the original DiLoCo work, Charles et al. (2025) study the performance of DiLoCo as model size and training data are scaled up. Crucially, the authors find that at each scale, DiLoCo's performance degrades as the number of workers is increased, but that as the model scale is increased, the performance degradation decreases. We complement these findings in our work, by showing that MuLoCo improves on worker scaling over DiLoCo relative to their respective data-parallel baselines and that at large worker counts, MuLoCo also improves with scale.

3.2 Distributed optimizers with compressed communication

Beyond leveraging infrequent communication, many distributed optimizers directly reduce the number of bits communicated through compression. This is typically done by sparsifying (Stich et al., 2018; Shi et al., 2019; Peng et al., 2024), quantizing (Alistarh et al., 2017), sketching (Rothchild et al., 2020), computing low-rank approximations of the gradient (Zhao et al., 2024; Vogels et al., 2019; Ahn et al., 2025), or combining any number of these approaches (Wang et al., 2023b). Among these works, many leverage error feedback for mitigating the bias introduced by lossy compression (Karimireddy et al., 2019).

Some works combine local SGD or DiLoCo with various compression methods. In the context of image classification, Basu et al. (2019) apply top-k sparsification and quantization to error-corrected moving averages of parameter deltas. Streaming DiLoCo (Douillard et al., 2025) combines partitioned communication, overlapped communication, and quantization to reduce communication cost in DiLoCo. In concurrent work, SparseLoCo (Sarfi et al., 2025) demonstrates that using a local error feedback as an outer momentum allows reaching extreme compression ratios in DiLoCo without compromising performance.

3.3 Single worker DiLoCo

Charles et al. (2025) established that the performance of DiLoCo with only a single worker is consistently stronger than AdamW alone. This is partially a rediscovery of the lookahead optimizer (Zhang et al., 2019b) and reptile (Nichol et al., 2018; Abbes et al., 2025), except that single-worker DiLoCo uses a Nesterov-style update as opposed to vanilla SGD. In follow-up work, Kallusky et al. (2025) establishes that compared to lookahead, single-worker DiLoCo (or SNOO) performs substantially better. They also show that the performance of single-worker DiLoCo when training MoE and dense LLMs can lead to a substantial speedup

over AdamW alone. In concurrent follow-up work, Defazio et al. (2025) propose GPA, an algorithm that generalizes and improves single-worker DiLoCo by eliminating the outer optimizer and enabling smooth iterate averaging at every step. GPA is shown to outperform single-worker DiLoCo while requiring one less accumulator. In this work, we establish the performance of single-worker MuLoCo at H=30 inner steps, showing that it outperforms data parallel Muon and has larger critical batch sizes. To the best of our knowledge, we are the first to establish K=1 MuLoCo's larger critical batch sizes and show that it obtains a better performance-training-time tradeoff than DP Muon. We expect that the performance improvements we establish will directly apply to GPA with a Muon inner optimizer.

3.4 Large-Batch Training and Critical Batch Size

Scaling batch size may be the simplest way to reduce training time, whether it be from increased data parallelism or improved hardware utilization. However, increasing batch size beyond a certain threshold, know as the critical batch size (CBS), leads to significant diminishing returns in performance. As such, many works have studied the large-batch training regime and its performance differences to using smaller batch sizes. Early empirical work highlighted that large batches can alter optimization dynamics and lead to poorer generalization behaviour (Keskar et al., 2017; Jastrzebski et al., 2017). At the same time, practical recipes such as linear learning-rate scaling and warmup demonstrated that large-batch training can match small-batch accuracy when carefully tuned (Goyal et al., 2017). Other work found that the generalization gaps found in earlier work stemmed from insufficient hyperparameter tuning, rather than inherent limitations of batch size (Shallue et al., 2019). In the context of language modeling, McCandlish et al. (2018) introduced the *gradient noise scale* as a predictor of the CBS, hypothesizing that the CBS coincides with the point at which increasing batch size no longer meaningfully reduces noise in the gradient. Recent work has revisited and refined this perspective. Zhang et al. (2025) demonstrate that critical batch size scales primarily with dataset size rather than model size. Merrill et al. (2025) propose a direct empirical measurement of the critical batch size, while Bergsma et al. (2025) derive scaling laws relating optimal and critical batch sizes to dataset size, providing an alternative methodology for predicting the CBS. Finally, several works establish that the critical batch size depends on the optimization algorithm (Zhang et al., 2019a; Lin et al., 2020). For instance, in the LLM regime, Vyas et al. (2025) show empirically that second-order preconditioned methods such as SOAP tolerate larger batches than AdamW, while Shah et al. (2025b) demonstrate that Muon can also reach larger critical batch sizes. Our study further expands on this, showing that MuLoCo achieves a Pareto optimal performance-batch size trade-off among all optimizers in our study (notably including DP Muon).

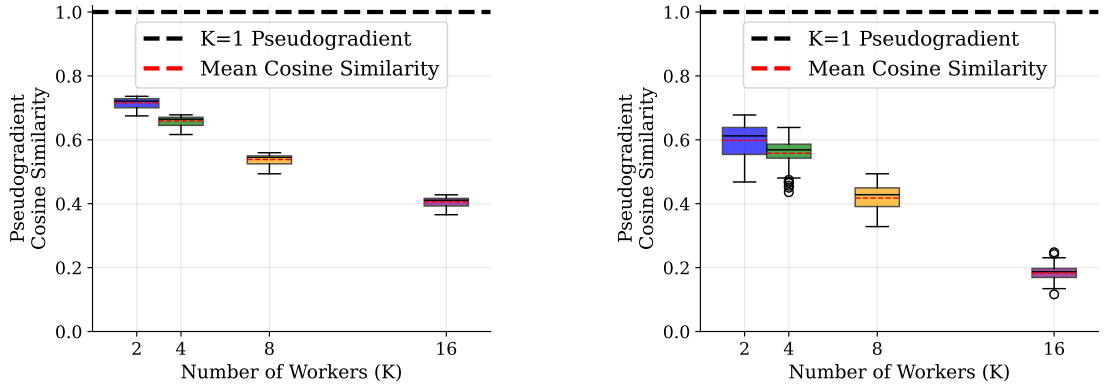
4 Method

Our main methodological contribution is to replace DiLoCo's AdamW inner optimizer with Muon. While this is a simple change, our results (Sec. 6) will show that beyond improving over DiLoCo in absolute terms, MuLoCo improves performance at larger worker counts over DiLoCo *even when normalizing by data-parallel Muon and AdamW, respectively.* That is, MuLoCo's improvements over DiLoCo **can not** entirely be explained by Muon's performance improvement over AdamW. Our main assumption is that this improvement can be explained by Muon's pseudogradients being more directionally correct than AdamW's. In this section, we introduce MuLoCo and theoretically analyse how the alignment of AdamW and Muon optimizer steps affects the pseudogradient's nuclear norm, offering a speculative explanation for the observed performance difference between MuLoCo and DiLoCo.

4.1 MuLoCo

MuLoCo reduces communication by training multiple models in parallel and infrequently communicating weight-space parameter differences after H steps of local optimization. Specifically, each worker $k \in [K]$ samples data from its assigned shard $\boldsymbol{D}_k$ and performs local optimization as follows:

$$\boldsymbol{\theta}_k^{(t+h)} = \mathrm{Muon}\left(\boldsymbol{\theta}_k^{(t+h-1)}, \nabla_{\boldsymbol{\theta}}\mathcal{L}\right), \quad h = 1, \ldots, H. \tag{1}$$



(a) MuLoCo **(b)** DiLoCo

Figure 2 Pseudogradient cosine similarities compared to a FLOP-matched $K = 1$ pseudogradient. Subfigure (a) shows results for Muon, while subfigure (b) shows results for AdamW. The box plots report the spread of cosine similarities across all hidden weight matrices of a 416M transformer. We observe that Muon's pseudogradients are more aligned with the K=1 pseudogradient, and this alignment decreases more slowly as the number of workers is increased.

The worker-level parameter differences, $\Delta_k^{(t)} = \boldsymbol{\theta}^{(t-H)} - \boldsymbol{\theta}_k^{(t)}$, are then communicated to form a pseudogradient:

$$\Psi^{(t)} = \frac{1}{K} \sum_{k=1}^{K} \Delta_k^{(t)}, \tag{2}$$

which captures the average weight-space displacement across all workers' inner optimization steps. The pseudogradient $\Psi^{(t)}$ is then used as input to SGD with Nesterov momentum, which updates its momentum and model weights as:

$$\begin{aligned} u^{(t)} &= \mu\, u^{(t-H)} + \eta_{\text{out}}\, \Psi^{(t)}, \\ \boldsymbol{\theta}^{(t)} &= \boldsymbol{\theta}^{(t-1)} - \mu\, u^{(t)} - \eta_{\text{out}}\, \Psi^{(t)}. \end{aligned} \tag{3}$$

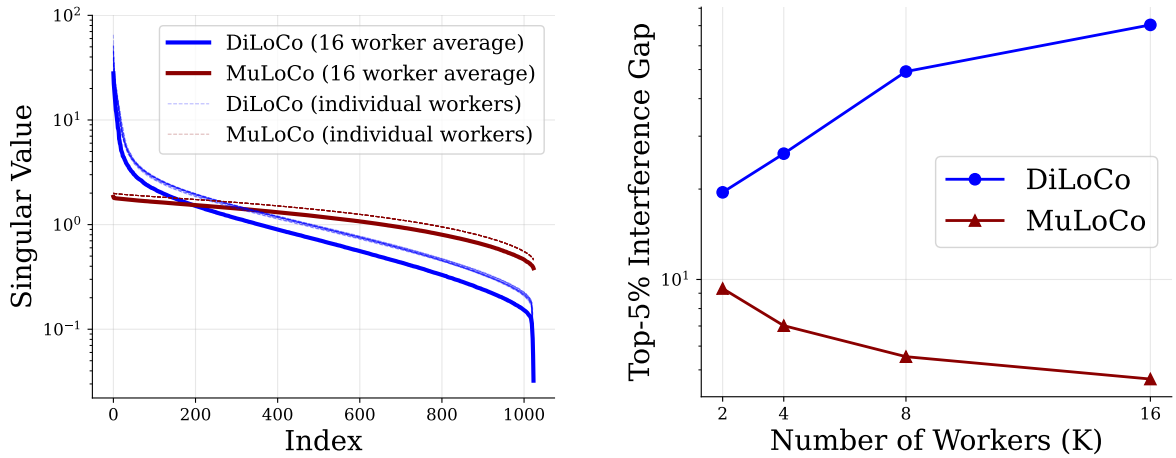
It should be noted that DiLoCo can be recovered by swapping Muon for AdamW in equation 1. Therefore, the pseudogradients of each method depend on the structure of individual optimizer updates. With this in mind, we will now theoretically connect the alignment of individual optimizer steps to the pseudogradient's nuclear norm, showing how they differ for Muon and AdamW.

4.2 Analyzing the benefits of Muon pseudogradients over AdamW

Our results (Sec. 6) will show that beyond improving over DiLoCo in absolute terms, MuLoCo improves performance at larger worker counts over DiLoCo even when normalized by its own data-parallel baseline. To better understand the reasons for this improvement we ran a number of analytical experiments targeted towards analyzing the impact of Muon and AdamW on pseudogradient quality, which Section 6.1 describes in depth. In the remainder of this section we will summarize the important results as they relate to our theoretical analysis.

Figure 2 reports the average cosine similarity across all hidden layers in a 12-layer transformer (416M in Table 1) for DiLoCo and MuLoCo at $K \in \{2, 4, 8, 16\}$ workers. The cosine similarity is computed between vectorized tensors and their corresponding K=1 pseudogradients. We find that at each worker count, MuLoCo's pseudogradient is more aligned with the K=1 pseudogradient (equivalent to the data-parallel weight difference). Moreover, the variance in cosine similarity across hidden tensors in the network is much larger for AdamW than Muon. These findings suggest that improved alignment may be a key factor in MuLoCo's improved performance at larger worker counts

In an attempt to better understand the exacerbated pseudogradient degradation at higher worker counts for DiLoCo relative to MuLoCo, in figure 3 (a) we visualize the pseudogradient spectra of DiLoCo and MuLoCo



(a) Pseudogradient Spectrum

(b) Top-5% Interference gap

Figure 3 DiLoCo's pseudogradient spectrum decays relative to the spectrum of its trajectories as the number of workers increases. (**a**) We plot the singular values of each worker weight difference Δ_k (dashed) and of the averaged pseudogradient Ψ (solid), where $\Psi = \frac{1}{K}\sum_{k=1}^{K}\Delta_k$. (**b**) We quantify destructive interference via the *top-S interference gap* $G_k = \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{S}\sigma_j(\Delta_k) - \sum_{j=1}^{S}\sigma_j(\Psi)$, i.e., the difference between the mean top-S Ky–Fan spectral mass of individual worker updates and the top-S spectral mass of their average. We observe that DiLoCo's interference gap grows as the number of workers increases, while Muon's shrinks, providing a possible explanation for DiLoCo's stronger performance degradation at large worker counts.

before and after averaging. We observe that the spectrum of the individual worker's pseudogradients collapse for AdamW during averaging, while the collapse is much less pronounced for muon. To measure this we introduce the interference gap.

Definition 4.1 (top-S interference gap)**.** Let $A_1, \ldots, A_N \in \mathbb{R}^{m\times n}$ and define their mean

$$\bar{A} = \frac{1}{N}\sum_{i=1}^{N} A_i.$$

Denote by $\sigma_1(X) \geq \cdots \geq \sigma_r(X) \geq 0$ the singular values of a matrix $X \in \mathbb{R}^{m\times n}$ with $r = \min\{m, n\}$. For any $k \in \{1, \ldots, r\}$, the *top-S interference gap* is

$$G_S = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{S}\sigma_j(A_i) - \sum_{j=1}^{S}\sigma_j(\bar{A}).$$

Here, $G_S \geq 0$ measures the loss of top-S spectral mass when averaging the matrices: $G_S = 0$ indicates perfect alignment of dominant singular directions, while larger G_S implies stronger destructive interference.

Figure 3 (b) reports the top 5% interference gap observed for MuLoCo and DiLoCo at different worker counts, finding that G_S decreases with worker count for MuLoCo but not DiLoCo. These findings reveal differences in behaviour between AdamW and Muon pseudogradients, suggesting that top-S interference may be related to the observed performance improvements for MuLoCo.

4.3 Relating Muon and AdamW Pseudogradient Spectra to Per-step Alignment

Inspired by our empirical results of the previous section, which show a meaningful difference between DiLoCo and MuLoCo pseudogradients in terms of alignment with the K=1 pseudogradient and top-S interference during averaging, in this section, we theoretically link the two. Specifically, we show that the nuclear norm of MuLoCo and DiLoCo pseudogradients is directly related to their alignment with individual optimizer steps.

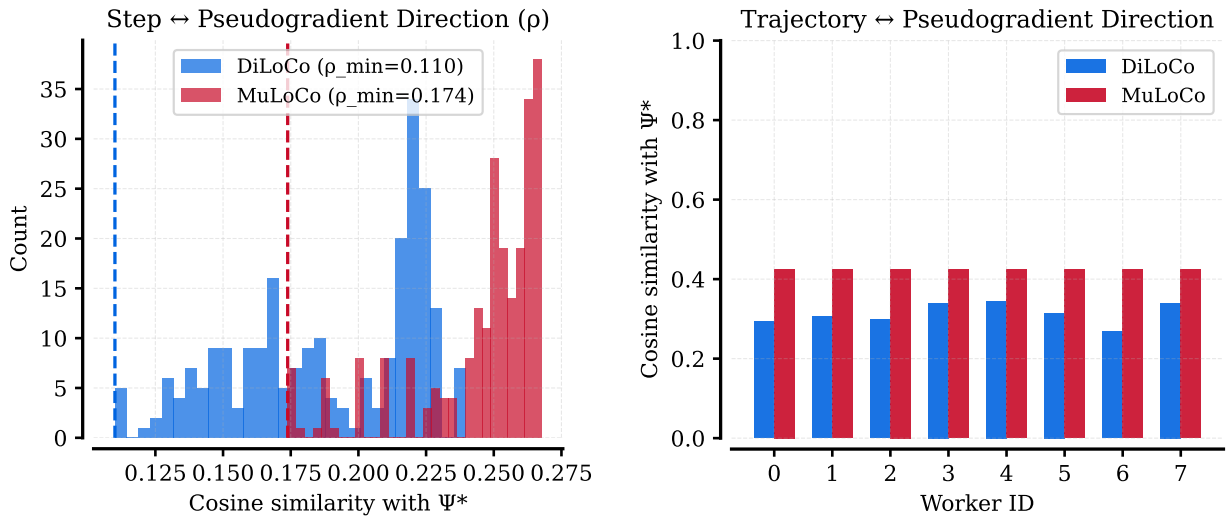


Figure 4 Alignment to the full pseudogradient. We report (left) the cosine similarity between *individual inner-optimizer steps* and the full communicated pseudogradient, and (right) the cosine similarity between *per-worker weight differences* (i.e., worker trajectories) and the full pseudogradient. We observe that MuLoCo's (Muon inner) optimizer steps are more consistently aligned with the final pseudogradient direction. Moreover, worker weight differences under Muon exhibit nearly identical cosine similarity to the final pseudogradient across workers, while AdamW shows considerably higher inter-worker variability.

AdamW inner updates and induced pseudogradient. For a given worker k and local step $h \in \{1, \ldots, H\}$ with gradient $g_k^{(t+h)}$, AdamW maintains first and second moments

$$m_{k,h} = \beta_1 m_{k,h-1} + (1-\beta_1) g_k^{(t+h)},$$
$$v_{k,h} = \beta_2 v_{k,h-1} + (1-\beta_2)\, g_k^{(t+h)} \odot g_k^{(t+h)},$$

and applies decoupled weight-decay leading to the following worker trajectory over H local steps

$$\Delta_{\text{AdamW}}^{(t,k)} = \sum_{h=1}^{H} \left(\alpha_{t+h} \frac{m_{k,h}}{\sqrt{v_{k,h}} + \varepsilon} + \lambda\, \alpha_{t+h}\, \boldsymbol{\theta}_k^{(t+h-1)} \right).$$

Muon inner updates and induced pseudogradient. With decoupled weight decay, Muon's orthonormalized update yield the following worker trajectory:

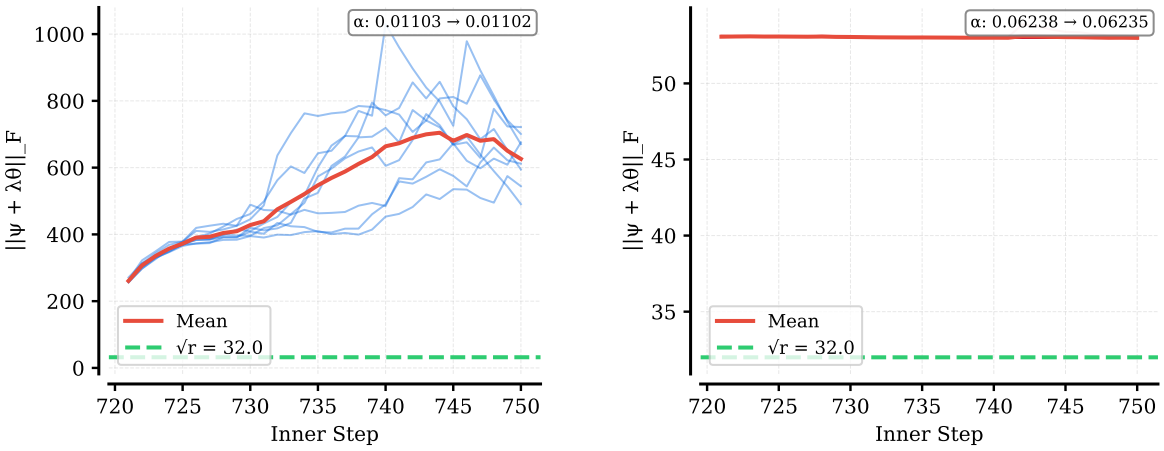
$$\Delta_{\text{Muon}}^{(t,k)} = \sum_{h=1}^{H} \left(\alpha_{t+h}\, O_{k,h} + \lambda\, \alpha_{t+h}\, \boldsymbol{\theta}_k^{(t+h-1)} \right).$$

For a more detailed explanation of Muon's update, we refer the reader to section 2.

Since the weight decay hyperparameter for Muon and AdamW is generally chosen to be $\lambda \leq 10^{-1}$ (this is the case in all our experiments), the contribution of decoupled weight decay to the pseudogradient is negligible. As such, we exclude it from our analysis. We now demonstrate that the nuclear norm of general pseudogradients depends on the alignment of individual optimizer steps with the pseudogradient and on the Frobenius norm of the optimizer step, itself.

Proposition 4.2 (Pseudogradient nuclear norm depends on optimizer step alignment)**.** *Fix $m, n \in \mathbb{N}$ and let $r := \min\{m, n\}$. Consider a pseudogradient of the form*

$$\Psi \;=\; \frac{1}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \alpha_{t+h}\, \psi^{(t+h,k)} \in \mathbb{R}^{m \times n}, \qquad \alpha_{t+h} \geq 0.$$



(a) Frobenius norm of **DiLoCo** inner steps. **(b)** Frobenius norm of **MuLoCo** inner steps.

Figure 5 MuLoCo's orthonormalized inner steps lead to stable Frobenius norms. Light blue (DiLoCo) and light red (MuLoCo) lines correspond to the Frobenius norm of the inner-optimizer step on each individual worker, while the red line reports their mean (MuLoCo's mean overlaps the individual workers). We observe that AdamW's per-worker step norms are comparatively erratic across workers, whereas Muon's are stable. Muon's step Frobenius norms are larger than the predicted $\sqrt{r}$ magnitude (for rank r) due to the additional contribution of decoupled weight decay.

Let $\Psi = U\Sigma V^\top$ be its singular value decomposition and define the orthonormal factor $\Psi^\star := UV^\top$. Let the cosine similarity between a given optimizer step and $\Psi^\star$ be

$$\rho^{(t+h,k)} := \frac{\langle \psi^{(t+h,k)}, \Psi^\star \rangle_F}{\|\psi^{(t+h,k)}\|_F \, \|\Psi^\star\|_F}.$$

Then the nuclear norm of Ψ satisfies

$$\|\Psi\|_* = \frac{\sqrt{r}}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h} \, \|\psi^{(t+h,k)}\|_F.$$

Proof. The proof is provided in section B of the appendix. □.

Corollary 4.3 (Muon pseudogradient)**.** *In the setting of Proposition 4.2, assume that the inner-step optimizer updates, $\psi^{(t+h,k)}$, are produced by Newton Shulz Iteration, making them orthonormal; then*

$$\|\Psi\|_* = \frac{r}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h}. \tag{4}$$

Proof. This follows from proposition 4.2 and the fact that $\psi^{(t+h,k)}$ is orthonormal, since $\|\psi^{(t+h,k)}\|_F = \sqrt{r}$. □

Corollary 4.4 (AdamW pseudogradient)**.** *In the setting of Proposition 4.2, for general inner-step matrices $\psi^{(t+h,k)}$ (e.g., AdamW steps), one always has*

$$\|\Psi\|_* = \frac{\sqrt{r}}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h} \, \|\psi^{(t+h,k)}\|_F. \tag{5}$$

Proof. The proof is identical to proposition 4.2. □

We observe that Muon's orthogonalized optimizer step results in the pseudogradient's nuclear norm naturally varying with the per-step alignment (ρ), the number of workers (K), and the inner learning rate (α). In contrast, the spectrum of AdamW's pseudogradient also depends on $\|\psi^{(t+h,k)}\|_F$, which may be highly variable

at higher worker counts, leading to overemphasizing certain steps unnecessarily. This exact behaviour is illustrated in Figure 5 where we empirically measure the Frobenius norm of individual optimizer steps, forming the pseudogradient for DiLoCo and MuLoCo. Subfigure (a) shows DiLoCo's pseudogradient Frobenius norm is highly variable across different workers at each inner step, while subfigure (b) shows that MuLoCo's is constant throughout. Intuitively, Muon's orthonormalized optimizer steps allow each individual optimizer step to contribute equally to the pseudogradient's spectrum. While it should be noted that our theory has no direct connection to performance and that all inferences made here are purely speculative, the slower decrease in cosine similarity of MuLoCo in Figure 2 relative to DiLoCo is correlated with its relatively stronger performance at larger worker counts (Figure 1 (a)). Together with our theory connecting the pseudogradients nuclear norm to the cosine similarity and Frobenius norm of individual optimizer steps, we speculate that the stronger alignment of MuLoCo's optimizer steps and its stable Frobenius norm across them may lead to a larger pseudogradient spectrum with less interference during averaging (e.g., as observed in Figure 3), which may improve performance.

5 Empirical Evaluation

Our goal is to establish the performance of MuLoCo relative to DiLoCo and their data-parallel baselines in the context of LLM pre-training. In Section 6 we start by studying various settings relevant to communication-efficient training on a small 416M language modelling task. Subsequently, in Section 7 we study scaling laws (150M-3B) of models pre-trained with these optimizers and scale up to $15B$. We now detail the exact experimental setting for these experiments.

Base LM pre-training Task. For all experiments except scaling laws, we select a 416M dense transformer language modeling task (see 416M in Table 1 for exact model dimensions). The architecture follows Gemma3 (Kamath et al., 2025) with SwiGLU FFNs, QK norm, and additional RMS normalization layers before residual connections. We tokenize using the Llama3 tokenizer Llama-Team (2024). Unless otherwise specified, we synchronize every $H = 30$ local steps and use $K = 8$ workers. In all of our experiments, we use a sequence length of 2048, we decay the learning rate following a cosine decay schedule to $0.1\times$ the maximum value, and we pre-train on the high-quality portion of the Nemotron-CC dataset (Su et al., 2024). For experiments scaling the communication steps H, with compressed updates, or using streaming DiLoCo, we fix the global batch size to $B = 1$M tokens. In our scaling experiments, the batch size is swept as described below.

Scaling study. In our scaling experiments, we increase model and dataset size following a ratio of 20 tokens per parameter (TPP) (Hoffmann et al., 2022). We report exact model sizes and token budgets in Table 1. At each scale except 15B, we conduct extensive hyperparameter sweeps as described below.

Table 1 Gemma3-style transformers used in our scaling study. The architecture follows Gemma3 (Kamath et al., 2025) with SwiGLU FFNs, QK norm, and additional RMS normalization layers before residual connections. However, we tokenize using the Llama3 tokenizer (Llama-Team, 2024). All models use a sequence length of 2048.

Model Parameters	Transformer Layers	Attention Heads	QKV Dimension	Hidden Dimension	Token Budget	HP Sweep
150M	6	4	512	1,408	3B	✓
416M	12	8	1,024	2,816	8.16B	✓
914M	18	12	1,536	4224	18.12B	✓
1.76B	24	16	2,048	5,632	35.23B	✓
3.07B	30	20	2,560	7,040	61.4B	✓
15.23B	54	36	4,608	12,672	304.6B	×

Hyperparameter tuning. At all model scales, we tune the weight decay (λ), inner learning rate (η_{in}), global batch size (B), outer learning rate (η_{out}), and outer momentum (μ) according to the following procedure.

1. (Data-Parallel λ) We first perform a grid search across λ within$\{1^{-1}, 1^{-2}, 1^{-3}, 1^{-4}\}$ and η_{in} values that are integer powers of $\sqrt{2}$ for the AdamW and Muon data-parallel baselines. We fix the $B = 1$M at all scales for this initial sweep.

2. (Data-Parallel η_{in}, B) Having established the optimal λ^* value at B =1M, we now perform a grid-search over integer powers of 2 for B and integer powers of $\sqrt{2}$ for η_{in}, while rescaling λ^* according to Wang and Aitchison (2024) as B is varied.

3. (DiLoCo/MuLoCo $\lambda, \eta_{\text{in}}, B$) Re-using the existing λ^* found above for B =1M, at each worker count $K \in \{1, 2, 4, 8, 16\}$ we perform a grid-search over powers of 2 B and powers of $\sqrt{2}$ η_{in}, while rescaling λ^* according to Wang and Aitchison (2024) *using the per-worker batch size (B/K).*

4. (DiLoCo/MuLoCo η_{out}, μ) When sweeping the outer optimizer's hyperparameters, we follow empirical guidance from Charles et al. (2025) who found that for DiLoCo they remain roughly constant across model sizes. As such, we perform an extensive grid search at the 416M scale sweeping across λ, η_{in}, η_{out}, μ, and B. During this extensive grid search, we tune η_{out} and μ by searching within $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ with other hyperparameters spanning ranges from above. We report the optimal μ and η_{out} values found in Figure 22 and re-use them at larger model scales.

DiLoCo We use AdamW as a baseline inner optimizer for DiLoCo. Following Charles et al. (2025) we set $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for all experiments. Unless otherwise specified, we use a synchronization interval of $H = 30$.

MuLoCo We use the Muon with a quintic Newton-Shulz iteration (Jordan et al., 2024). Muon is applied to hidden layers, while AdamW is used for the embeddings, normalization, and output layers. For hidden matrices of size $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ we rescale the learning rate by $\sqrt{\frac{n}{m}}$. We set $\beta_1 = 0.9$ for both Muon and AdamW. We set AdamW's $\beta_2 = 0.99$. Unless otherwise specified, we use a synchronization interval of $H = 30$.

Outer Optimizer We use SGD with Nesterov momentum as the outer optimizer for all our DiLoCo and MuLoCo experiments with hyperparameters reported in Figure 22.

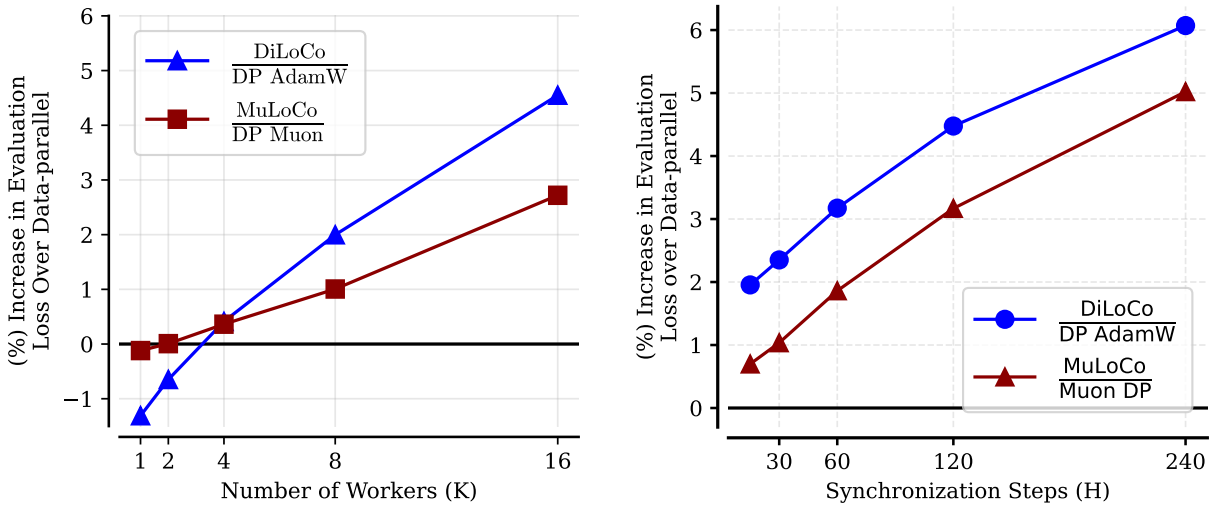
Data Parallel Baselines We also report the performance of well-tuned DP AdamW and DP Muon baselines.

Implementation For Muon, we use Ahn et al. (2025)'s distributed implementation. For AdamW, we used the fused PyTorch implementation. All our pre-training experiments are run using TorchTitan (Liang et al., 2025). We use the streaming DiLoCo implementation within TorchFT for all our distributed optimizers. We train across H100 and H200 GPUs. Our largest experiments at 15B scale use HSDP with 16 FSDP ranks and 64 data-parallel ranks. For DiLoCo and MuLoCo, they are accordingly subdivided among workers.

Evaluation Loss During training, we log the validation loss every 15 steps on 2M token batches from a held-out set of Netotron-CC pre-training data. Following (Hoffmann et al., 2022), to reduce stochasticity in the final loss estimate, we use a smoothed final loss. Specifically, we first subsample the validation loss trajectory to synchronization boundaries (every $H = 30$ steps) and apply a time-weighted EMA with base smoothing parameter $\alpha = 0.2$. At the nominal spacing of $H = 30$ steps, the adaptive coefficient is $\tilde{\alpha} \approx 0.181$, corresponding to an effective smoothing window of approximately 5–6 synchronization rounds. This procedure yields a stable final loss estimate that is less sensitive to variance across validation batches. The resulting $\hat{L}$ is used consistently across all analyses in this paper, including hyperparameter selection, scaling law fitting, and cross-method comparisons. Further details are reported in Section F of the appendix.

6 Results—Communication Efficient Ablations & Wall-Clock Time

The following section compares the performance of MuLoCo and DiLoCo at 416M scale across a number of settings relevant to communication-efficient training: (6.1) training across a large number of workers ($K = 1 - 16$), (6.2) training with long synchronization intervals ($H = 15 - 240$), (2) compressing the pseudogradient (via quantization and sparsification), (6.4) reducing peak bandwidth with partitioned communication from streaming DiLoCo, and (6.5) when accounting for wall-clock training time.



(a) Scaling workers (K). **(b)** Increasing synchronization interval (H).

Figure 6 MuLoCo consistently outperforms DiLoCo, including relative to their respective data-parallel (DP) baselines at larger worker counts. Subfigure (a) (same as subfigure (a) in Fig. 1) reports the percentage increase in evaluation loss for DiLoCo and MuLoCo relative to their DP baselines. We observe that the relative performance of each method degrades as K is increased, but MuLoCo shows clear improvements over DiLoCo *relative to* their data-parallel baselines as the number of workers grows. **Subfigure (b)** varies the synchronization interval H and shows that MuLoCo continues to outperform DiLoCo relative to their DP baselines, maintaining MuLoCo's improvements at K=8 workers across different synchronization intervals.

Takeaways:

- **Better worker scaling.** beyond outperforming DiLoCo in absolute terms, MuLoCo outperforms DiLoCo at $K > 2$ workers *even* when normalizing by their respective DP baselines.
- **Compatible with reduced communication.** MuLoCo outperforms DiLoCo under long synchronization intervals, pseudogradient compression, and when using with partitioned communication.
- **Better memory complexity.** MuLoCo has a lower memory complexity than DiLoCo (3X v.s. 4X).
- **Low optimizer step overhead.** Despite having a more expensive optimizer step, we find that MuLoCo's overhead in training time is relatively low at less than 1%. Despite this, we find that it has better wall-clock training times due to its improved optimization.

6.1 MuLoCo vs DiLoCo: Increasing K

Figure 6 (a) reports the increase (lower is better) in final validation loss relative to data-parallel as a function of the number of workers (k). It should be noted that on an absolute scale, MuLoCo strictly outperforms DiLoCo in each case due to Muon's stronger performance. However, we also observe that both DiLoCo and MuLoCo outperform their respective data-parallel baselines for $K = 1$ and $K = 2$ workers. At these worker counts, DiLoCo's improvements over data-parallel are larger than MuLoCo's, suggesting that Muon's curvature-aware steps may reduce the gains available from using an outer momentum. For $K > 2$ workers, we see a clear pattern emerge: *as the number of workers increases the performance of DiLoCo relative to its data parallel baseline decreases at a faster rate than MuLoCo does relative to Muon DP, demonstrating that MuLoCo's performance scales more favourably as workers are increased.*

Finding 1: Relative to their data-parallel baselines, MuLoCo scales better than DiLoCo as the number of workers increases. (Fig. 1 (a))

Why does MuLoCo scale better with workers? Our central hypothesis is that Muon's orthonormalized optimizer steps lead to pseudogradients that are more directionally aligned with the data-parallel/K=1 pseudogradient,

allowing MuLoCo to more accurately approximate it at higher worker counts. To validate this, we measure the alignment of MuLoCo's and DiLoCo's pseudogradients with their data-parallel/K=1 pseudogradients. To accomplish this, we use the optimal hyperparameters for DP Muon and AdamW found in our sweeps and train a 416M model saving checkpoints along the way. We subsequently resume training from these checkpoints using DiLoCo or MuLoCo with $K \in \{2, 4, 8, 16\}$, including loading optimizer states, and continue to train using the same hyperparameters and global batch size for H=30 steps, after which we save pseudogradients across all workers to disk. We then measure alignment of pseudogradients at $K > 1$ with the DP baseline/K=1 pseudogradient. It is calculated by measuring the consine similarity between all vectorized hidden weights in the network and taking the average of these values.

Figure 2 reports the cosine similarity as a function of worker count for MuLoCo (a) and DiLoCo (b). It clearly demonstrates that MuLoCo's pseudogradients are consistently more directionally aligned with the the data-parallel/K=1 pseudogradient than DiLoCo. This suggests that Muon's inner steps may play a crucial role in effectively shaping the pseudogradient. Pursuing this hypothesis further, in Figure 4 (a), we report the cosine similarity between a $K = 8$ pseudogradient (collected as described above) and the individual optimizer steps that make it up. We find that Muon's optimizer steps are much more aligned with the final pseudogradient direction than AdamW's, suggesting that *Muon optimizer steps are naturally more aligned with one another*. Thus far, we have provided evidence of MuLoCo's more *directionally correct* pseudogradients but we have not explained why DiLoCo's pseudogradients degrade at larger worker counts. We hypothesize that the poorer alignment of AdamW steps may lead to exacerbated information loss during averaging. In an effort to better understand this, we investigate the spectrum of individual worker weight differences Δ_k before and after averaging them to form a pseudogradient. In Figure 3 we show that DiLoCo's pseudogradient spectrum collapses during averaging to a much greater extend that MuLoCo's and that the collapse worsens at larger worker counts. This is corroborated by DiLoCo's relatively stronger performance at K=1 and K=2 workers, when interference is low, with a sharp degradation thereafter. While they do not show a direct causal link, *these findings are consistent with our hypothesis that information loss during averaging is the main mechanism by which DiLoCo pseudogradients degrade.*

Why does K=1 DiLoCo perform better relative to DP AdamW than MuLoCo does relative to DP Muon? Despite always outperforming K=1 DiLoCo in absolute terms, in Figure 6 (a) K=1 DiLoCo's improvement is larger than K=1 MuLoCo relative to their respective data-parallel baselines. We hypothesize that this is due to the greater alignment with the final pseudogradient observed for Muon optimizer steps than for AdamW's (Fig. 4). The outer optimizer in DiLoCo/MuLoCo is analogous Reptile meta-learning, which is known to encourage alignment between different optimizer steps (Nichol et al., 2018). Therefore, since Muon's optimizer's steps are already well aligned, it stands to reason that the outer optimizer would be less effective at improving its performance than for AdamW.

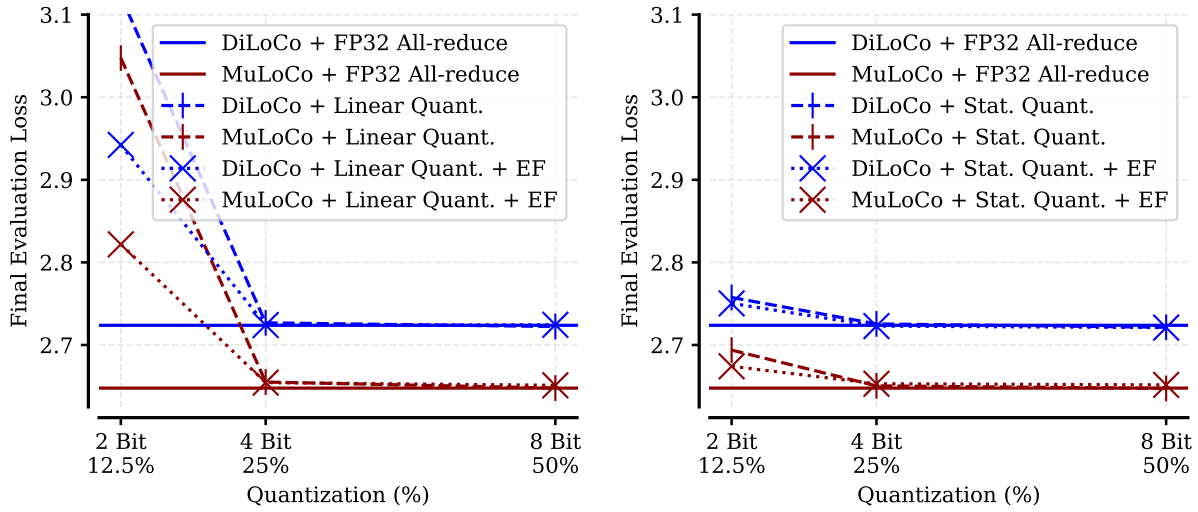
6.2 MuLoCo vs DiLoCo: Increasing H

In this section, we study the performance of MuLoCo and DiLoCo relative to their data-parallel baselines as the synchronization interval (H) is progressively doubled from $H = 15$ to $H = 240$. To accomplish this, we train DP baselines in our chinchilla-optimial 8B token setting for 8160 steps at batch size 512 and sequence length 2048. For DiLoCo and MuLoCo, we fix the number of workers at $K = 8$ and vary H. Figure 6 (b) reports the performance of both methods relative to data-parallel. We observe that for all synchronization intervals tested, MuLoCo outperforms DiLoCo relative to their respective data-parallel baselines. Furthermore, the improvements of MuLoCo over DiLoCo remain mostly constant across H values, showing that the communication frequency no effect on performance differences between DiLoCo and MuLoCo relative to DP.

> **Finding 2**: MuLoCo maintains a *relative* performance advantage over DiLoCo as the synchronization interval is increased. (Fig. 6 (b))

6.3 MuLoCo vs DiLoCo: Pseudogradient Compression

In this section, we evaluate the performance of MuLoCo and DiLoCo when compression is applied to further reduce communication costs. Specifically, we quantize or sparsify the parameter delta in line 15 of Algorithm 2 before communicating it and ablate using error feedback (Karimireddy et al., 2019).



(a) Linear quantization **(b)** Statistical quantization

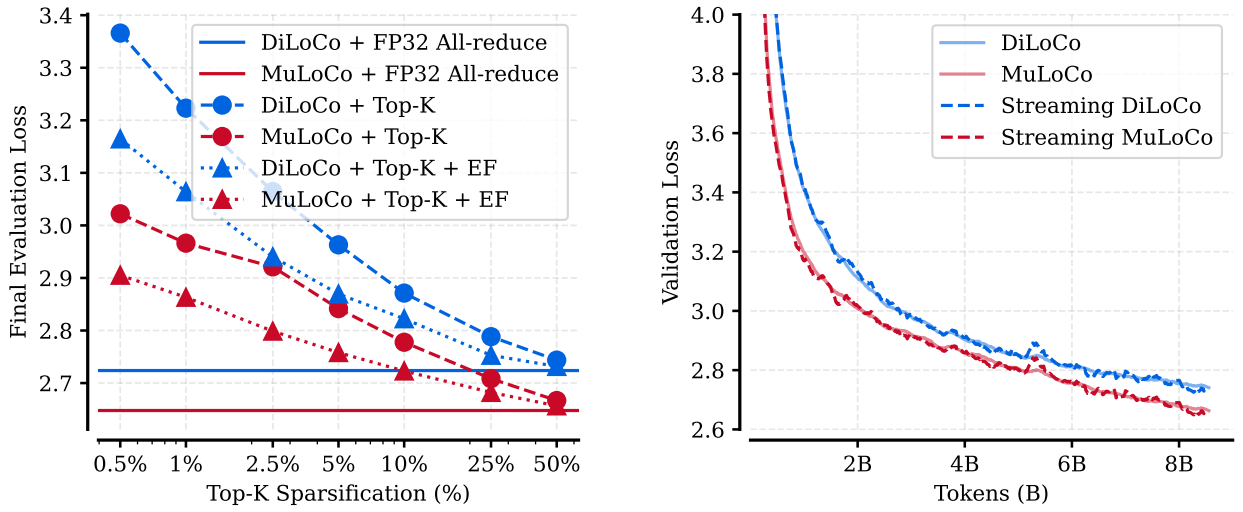
Figure 7 Quantized pseudogradient compression: linear vs. statistical quantization and the effect of error feedback (EF). Linear quantization (left) and row-wise statistical quantization (right) are shown with and without EF across bitwidths. In each case, MuLoCo outperforms DiLoCo. For 4 bits and above, both statistical and linear quantization behave essentially identically with or without EF, suggesting that EF is unnecessary in this regime. At 2 bits, statistical quantization is consistently better than linear, indicating that it preserves update quality under more aggressive quantization. Overall, EF provides only a modest improvement, except for 2-bit linear quantization, where EF can partially recover performance; however, the remaining degradation is still severe, making 2-bit linear quantization impractical in this setting.

Quantization Figure 7 reports the final evaluation loss of models trained using DiLoCo and MuLoCo with 8-bit, 4-bit, and 2-bit quantization in a K=8, H=30 setting. We experiment with global linear quantization (a) and statistical quantization (b) (see Fig. 15 for row-wise versions of each algorithm). Our experiments assume that communication is performed using an efficient all-to-all reduce-scatter (Wang et al., 2023a) and subsequent all-gather to avoid compounding quantization errors incurred by a ring all-reduce with many workers. We observe that MuLoCo outperforms DiLoCo in absolute terms and that both achieve lossless compression up to 4-bits. When compressing to 2-bits there is a substantial performance degradation for Linear quantization but not statistical. In Figure 15 we report additional experiments where quantization is applied individually to each row of a tensor. We find that this improves the performance of linear quantization at 2-bits and in the statistical case has near identical performance to global quantization.

Finding 3: Both MuLoCo and DiLoCo achieve lossless 4-bit quantization. (Fig. 7)

Global Quantization v.s. Row-wise quantization. With simple and efficient implementations in mind, we decided to study both global and row-wise quantization. Global quantization is simple to implement and has minimal metadata overhead; however, it does require computing statistics across the entire tensor, which can be inefficient for large models or when parameters are sharded. Row-wise quantization was chosen for its improved performance characteristics and its affinity for parallelization. While the row-wise approach has the requirement to store metadata for each row, this avoids the need to synchronize metadata and tensor value information during the reduction since everything can be sent at once. It is also amenable to highly parallelized dequantize-reduce-quantize operations on GPUs since each row can be processed independently.

Top-k Sparsification Figure 8 reports the performance of MuLoCo and DiLoCo with and without error-feedback under top-k sparsification retaining $0.5\%, 1\%, 2.5\%, 5\%, 10\%, 20\%$, and 50% of entries. We observe that MuLoCo consistently outperforms DiLoCo and that Error feedback helps prevent performance degradation. However, the degradation in performance can be substantial at larger sparsities. Given the additional overhead of communicating the index, the true compression ratios of vanilla top-k sparsification are higher than the reported sparsity, making the more competitive sparsity ratios of 5-10% less appealing than using 2-bit



(a) Top-k sparsification

(b) Streaming updates

Figure 8 Top-k compression and streaming. Left (Top-k): MuLoCo remains more robust than DiLoCo at higher sparsities (i.e., smaller communicated fractions), achieving lower final validation loss at aggressive sparsification levels. Error feedback (EF) improves both methods, but does not make extreme sparsities viable: performance degrades rapidly as sparsity increases. **Right (Streaming):** Streaming and non-streaming variants match closely for both DiLoCo and MuLoCo, indicating no degradation from partitioned (streamed) communication. In particular, MuLoCo tracks DiLoCo under streaming and remains fully compatible with streaming updates.

quantization with an all-to-all reduce scatter. Nevertheless, if very strong compression is the goal, practitioners can significantly benefit from using MuLoCo with error feedback over DiLoCo.

6.4 Streaming MuLoCo

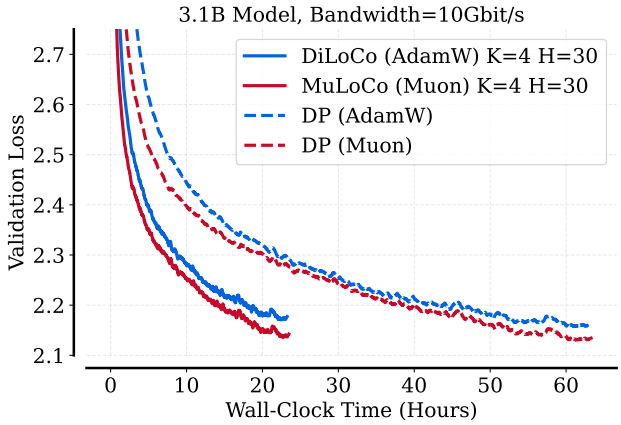
In this section, we investigate MuLoCo's performance when using partitioned communication (Douillard et al., 2025). Specifically, in our K=8 worker and H=30 communication step setting, we partition the model's parameters into thirds and take outer steps for partitions every 10 steps. This has the effect of reducing the peak communication volume by a factor equal to the number of partitions. Figure 8 (b) reports validation loss curves during training for DiLoCo, MuLoCo, and their streaming counterparts. We observe that both the streaming and non-streaming methods reach the same final validation loss, demonstrating that MuLoCo is compatible with streaming.

Finding 4: MuLoCo is compatible with the partitioned communication of streaming DiLoCo.

6.5 Wall-clock training time

In this section, we (1) validate the practicality of using Muon as the inner optimizer for DiLoCo, given its more expensive optimizer step, and (2) illustrate the utility of MuLoCo for training under bandwidth constraints. Using the distributed implementation of Ahn et al. (2025) in combination with HSDP within TorchTitan (Liang et al., 2025), we train a 3.1B model across four 8×H100 nodes using FSDP within nodes and data parallelism across nodes. Table 9 reports the end-to-end step time, MFU, TFLOPS/GPU, throughput, and training time in a chinchilla optimal regime. We find that the optimizer has negligible training time overhead: +0.96%. In figure 9 we report wall-clock training curves, demonstrating that Muon's overhead is negligible (MuLoCo dominates DiLoCo) and showing the significant benefits of using MuLoCo over Muon DP in a bandwidth-constrained environment with 10 Gbit/s networks (23 v.s. 60+ hours).

Finding 5: MuLoCo has negligible wall-clock overhead relative to DiLoCo. (Tab. 9)



Metric	DiLoCo	MuLoCo	Δ (%)
End-to-End Time (s)	2.82	2.85	+0.96
MFU (%)	43.04	42.63	−0.95
TFLOPS/GPU	425.7	421.7	−0.95
Tokens/s/GPU	23,250	23,029	−0.95
Training Time (H)	23.26	23.49	+0.96
Memory Complexity (Parameter copies)	4	3	−25

Figure 9 Wall-clock training and system-level performance comparison. Left: Idealized wall-clock training time under bandwidth constraints (10 Gbit/s) for MuLoCo, DiLoCo, and their DP baselines. DiLoCo completes training slightly faster due to its lower per-step overhead, but MuLoCo's faster convergence and better worker scaling lead to better performance at any training duration. Both methods reach lower loss much faster than their DP baselines due to communication efficiency. **Right:** System metrics (median over first 500 steps). Configuration: 3.1B model, 2M token batch (1024×2048), $K = 4$ workers, $H = 30$, 4×8 H100 GPUs. MuLoCo has negligible overhead ($< 1\%$) relative to DiLoCo.

7 Results—Scaling up

In this section, we investigate whether MuLoCo's performance improvements over DiLoCo hold as model and dataset size are increased under a chinchilla optimal prescription (Hoffmann et al., 2022). To accomplish this we extensively tune MuLoCo, DiLoCo, and their respective data parallel baselines on 150M-3.1B language model pre-training tasks (see Tab. 1) and fit power laws to their final evaluation loss. We use a communication interval of $H = 30$ steps for these experiments and do not use any compression or streaming. Hyperparameters of all optimizers are extensively tuned at each worker count, $K \in \{1, 2, 4, 8, 16\}$. These include the weight decay (λ), inner learning rate (η_{in}), global batch size (B), outer learning rate (η_{out}), and outer momentum (μ). We refer the reader to section 5 for further details of our hyperparameter tuning process.

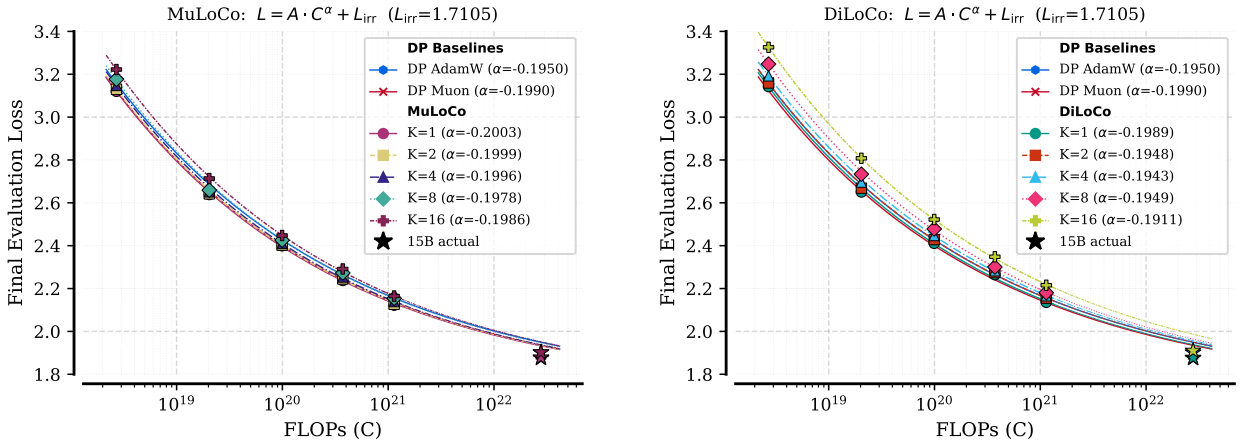
The following subsections present the data collected from this study, highlighting the most important results: (7.1) projected performance improvement of MuLoCo and DiLoCo at scale, (7.2) MuLoCo's larger optimal and critical batch sizes, and (7.3) evaluation results for training 15B models.

Takeaways:

1. **MuLoCo's improved worker scaling relative to DiLoCo is maintained at scale.** At $K > 2$ MuLoCo continues to outperform DiLoCo at scale *even* when normalized by their respective DP baselines.
2. **MuLoCo's performance improves relative to data-parallel with scale.** At K=16 workers, the performance gap between MuLoCo and DP Muon shrinks, suggesting that the former will approach the latter's performance at larger scales while being more parallelizable.
3. **MuLoCo has larger critical batch sizes (CBS) than DiLoCo.** Our results at 3.1B scale demonstrate that MuLoCo has a consistently larger CBS than DiLoCo at any K.
4. **At iso-loss, K=1 MuLoCo has better training-time-efficiency than DP Muon, K=1 DiLoCo, and DP AdamW.** Fitting scaling laws to performance at CBS, we find that K=1 MuLoCo requires, within observed data, up to 4.5× fewer sequential FLOPs (≈ training time) to reach the same loss as DP AdamW.

7.1 Extrapolating MuLoCo's performance when scaling compute.

In this section, we inspect whether MuLoCo's improved performance at higher worker counts extends to larger scales and whether MuLoCo's performance at larger worker counts improves with scale as was found



(a) MuLoCo scaling law fit in compute

(b) DiLoCo scaling law fit in compute

Figure 10 Power-law fits in compute with a shared irreducible loss. We fit power laws of the form $L(C) = A\,C^{\alpha} + L_{\text{irr}}$ to the final evaluation loss as a function of training compute C, using a *joint* (shared) irreducible loss L_{irr} across methods. The fitted exponents α are larger for MuLoCo than DiLoco, indicating better performance with scale. We also observe that the fit underestimates the final loss attained by our 15B runs.

for DiLoCo (Charles et al., 2025).

We first evaluate the fit of three candidate compute scaling laws: (i) $L(C) = aC^{\alpha}$, a simple power law; (ii) $L(C) = aC^{\alpha} + c$, with a per-optimizer irreducible loss; and (iii) $L(C) = aC^{\alpha} + L_0$, with a joint irreducible loss L_0 shared across all 12 optimizer/worker combinations while each retains independent a and b.

Fitting scaling laws. We fit the parameters of all scaling laws by minimizing a Huber loss (Huber, 1964) in log-space. Specifically, given predictions $\hat{L}i$ and targets L_i, we minimize $\sum_i H_\delta(\log \hat{L}i - \log L_i)$ where H_δ is the Huber loss function with $\delta = 0.001$. This provides robustness to outliers while remaining smooth near zero. Optimization is performed using L-BFGS (Nocedal, 1980) with a maximum of 15,000 iterations per run. To avoid poor local minima, each fit is repeated from 512 random initializations, and the solution with the lowest objective is retained. For joint irreducible loss fitting, a three-phase grid search is used: a coarse sweep over 200 candidate L_{irr} values (each evaluated with 64 restarts per combination), a zoom phase refining the neighborhood of the best candidate, and a final phase re-fitting all combinations at the selected L_{irr} with the full 512 restarts.

Table 2 Comparison of compute scaling law forms. We hold out the 3.1B scale from fitting for each power law. We report the mean absolute log-space residual across all 12 optimizer/worker combinations. We find that adding an irreducible loss substantially improves extrapolation to unseen scales.

Functional Form	Train Residual 150M–1.7B	Evaluation Residual 3.1B	15B
$L(C) = AC^{\alpha}$	0.013	0.056	0.160
$L(C) = AC^{\alpha} + c$	**0.001**	**0.011**	**0.053**
$L(C) = AC^{\alpha} + L_{\text{irr}}$	**0.001**	**0.011**	0.057

Evaluating functional forms for L(C). Table 2 reports residuals from the different scaling laws fit to $150M - 1.7B$ datapoints. We hold out 3.1B scale to evaluate the performance of each method. We find that adding an irreducible loss substantially improves extrapolation, reducing the mean 3.1B residual from 0.056 to 0.011. Turning to the 15B residuals, we observe that using an irreducible loss also improves extrapolation at this scale, but that the source of error changes: it leads to under-reporting 15B performance when $L(C) = aC^{\alpha}$ would have over-reported it. We also found that the extrapolation is marginally improved at 15B scale when fitting individual irreducible losses. However, given that 15B was not swept to optimality and that the assumption of a shared irreducible loss is a reasonable one, we opt to use the power law with a joint irreducible loss.

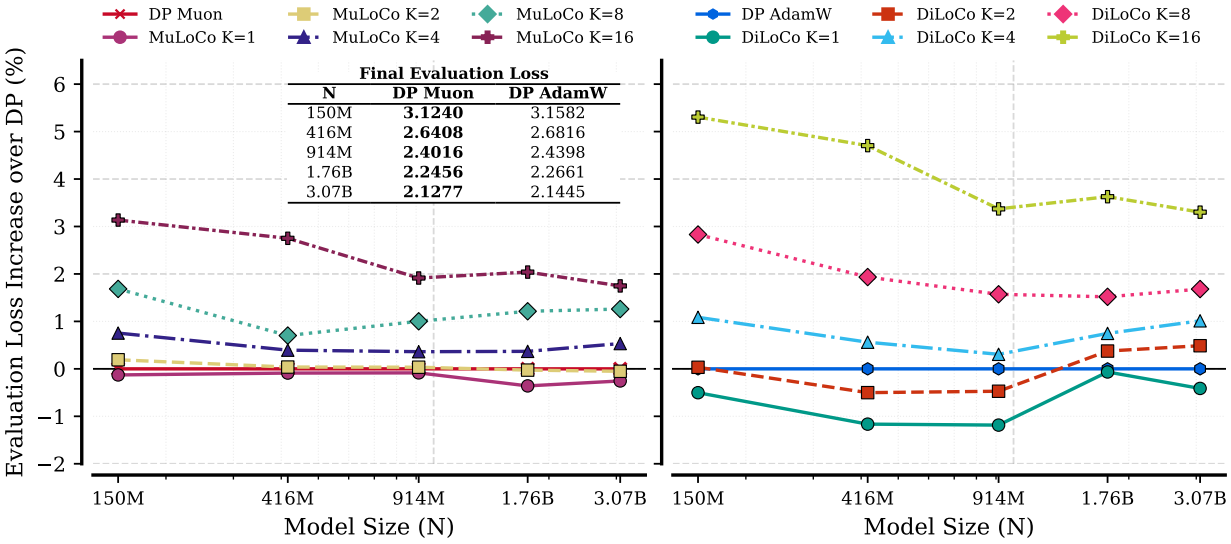


Figure 11 Relative performance vs. data-parallel baselines as model size and worker count scale. We report the percent increase in final evaluation loss over data-parallel (DP) for MuLoCo (left) and DiLoCo (right) across model sizes and worker counts $K \in \{1, 2, 4, 8, 16\}$. MuLoCo's performance relative to DP improves at larger worker counts, and we observe the relative gap shrinks with scale for $K = 16$ workers. Note that MuLoCo is normalized by *Muon* DP, while DiLoCo is normalized by *AdamW* DP.

Final L(C). We now re-fit the power law with a joint irreducible loss to all five training scales (150M-3.1B). Table 6 of the appendix reports the fitted parameters, while Figure 10 plots the corresponding curves. We find that an irreducible loss of $L_{\text{irr}} = 1.711$ achieves the best fit. The scaling exponents cluster around $\alpha \approx -0.20$ ($\alpha \in [-0.200, -0.191]$), with MuLoCo having generally larger α values than DiLoCo across the board, indicating stronger scaling with compute. When evaluating the extrapolation to 15B we observe that, as we found above, the power laws consistently underreport performance at larger scale, suggesting that an irreducible loss of $L_{\text{irr}} = 1.711$ is likely too large in practice. Given this finding, in figure 17 of the appendix, we fit compute scaling laws at fixed L_{irr} values shared across all methods. For each L_{irr}, we report the resulting scaling exponent α normalized by the exponent of the corresponding DP baseline. We find that at lower irreducible losses, which are likely to be more realistic than $L_{\text{irr}} = 1.711$, MuLoCo at higher worker counts is projected reach and eventually outperform data parallel baseline.

Finding 6: The scaling exponents of Muon-based methods are smaller, demonstrating their stronger scaling.

Performance at different worker counts. Figure 11 reports the performance of MuLoCo and DiLoCo normalized by their respective data parallel baselines as a function of model size. First, we find that performance improvements of MuLoCo over DiLoCo, even when normalized by their respective data-parallel baselines, hold at larger scales. Second, we interpret our results within the context of Charles et al. (2025) who found that, at all worker counts (K), DiLoCo follows a trend of strict performance improvement relative to DP AdamW as model size is increased. In our experiments, we find the same trend for K=16 MuLoCo and K=16 DiLoCo, suggesting that these optimizers should continue to improve relative to data parallel with scale. However, we don't observe the same strict trend of monotonic performance improvement with model size. While we are uncertain about the exact cause of this discrepancy, we conjecture that it may be caused by one or many of the following (listed in no particular order): differences in hyperparameter tuning, differences in pre-training data, differences in how evaluation loss was calculated, our choice to additionally tune the outer optimizer's momentum, our decision to decay the learning rate to 1/10, and other experimental discrepancies.

Finding 7: At K=16 MuLoCo consistently improves with scale relative to the DP Muon.

Finding 8: MuLoCo's improved performance over DiLoCo at $K > 2$ *even* when normalized by their respective data-parallel baselines is maintained at scale.

7.2 On MuLoCo's large critical batch size

Maintaining strong performance at large batch sizes is very important for efficiently training the largest models. Training at these batch sizes requires leveraging a larger number of devices in parallel, which can easily span poorly interconnected regions, such as across datacenters or across racks within a datacenter. In such situations, communication-efficient optimizers can be very beneficial. However, *a prerequisite is to be able to leverage large batch sizes to begin with.* Our results in this section will demonstrate that MuLoCo has significant benefits over DiLoCo in this regard.

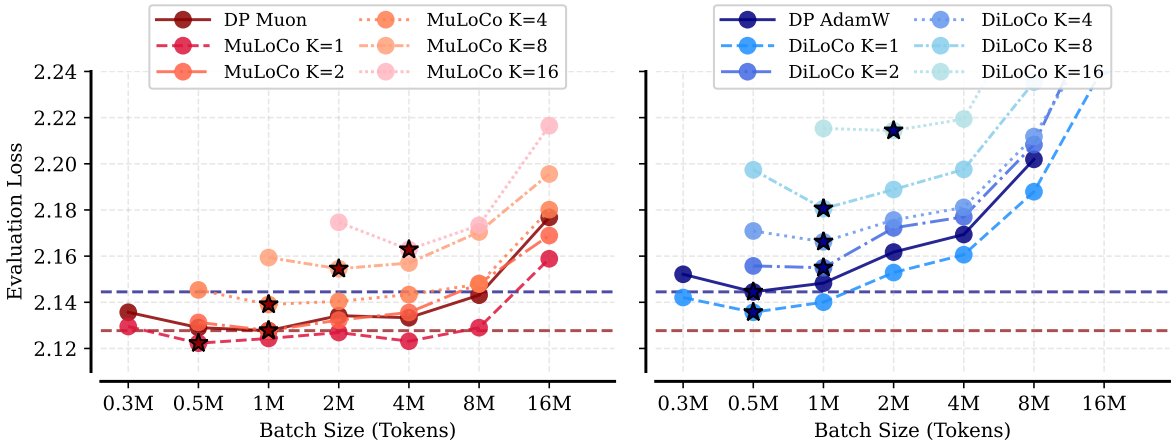


Figure 12 MuLoCo has much larger critical batch sizes than DiLoCo. We train 3.1B-parameter models and *tune hyperparameters separately at each batch size* for every method, reporting the *final evaluation loss* at the end of training. We observe that MuLoCo has larger optimal batch sizes at $K > 1$ and a much larger critical batch size at all worker counts, enabling substantially more parallelizable training (potentially large enough to require cross-datacenter training). In contrast, DiLoCo degrades rapidly as batch size increases, leading to a much smaller usable batch-size regime.

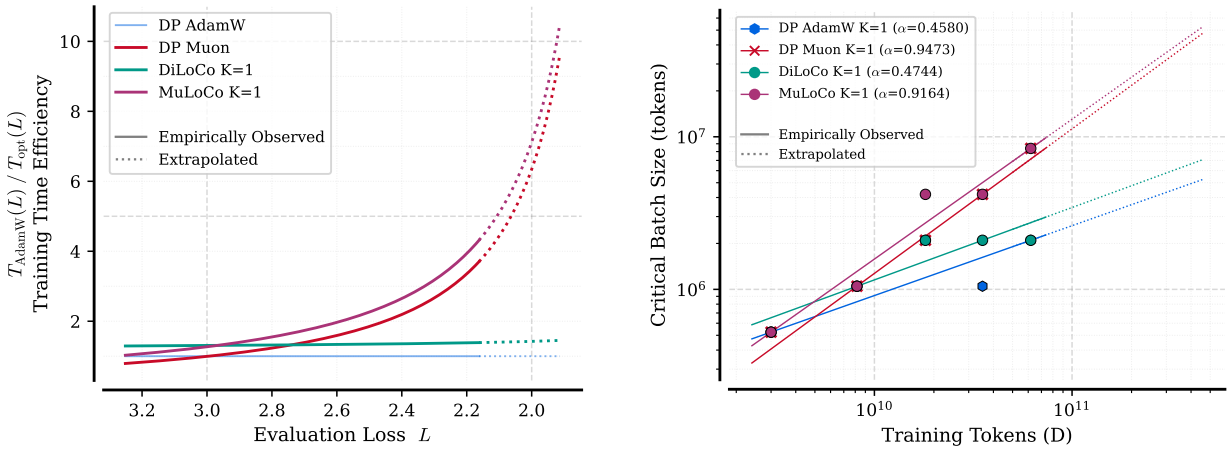
Critical batch size at 3.1B. Figure 12 reports the final evaluation loss of 3.1B models tuned at each batch size. We observe that AdamW degrades substantially as the batch size increases, while MuLoCo maintains strong performance relative to its optimal batch size until $\sim$ 8M tokens at all worker counts. MuLoCo's increased critical batch sizes result in increasingly parallelizable training. At the largest scales, taking full advantage of this parallelism to reduce training time will likely require training across datacenters, *leading to communication bottlenecks for DP Muon, but* **not** *for MuLoCo.* Even when communication is not a bottleneck, MuLoCo at K=1 still offers a notable advantage over Muon DP, maintaining stronger performance at larger batch sizes. This is particularly noticeable at $B = 8$M where MuLoCo K=1 matches the optimal performance of Muon DP, achieved at a much smaller $B_{\text{opt}} = 1$M.

Finding 9: MuLoCo's critical batch size at 3.1B scale is larger than DiLoCo's at all worker counts (K).

Extrapolating the Critical Batch Size. Given that the critical batch size is known to scale with training data (Bergsma et al., 2025; Zhang et al., 2025), we fit power laws $B_{\text{crit}}(D) = aD^{\alpha}$ to extrapolate the critical batch size (B_{crit}) as training data (D) is increased. With B_{opt} being the optimal batch size, we take B_{crit} to be the largest batch size such that $L(B_{\text{crit}}) \leq 0.01L(B_{\text{opt}})$. We exclude MuLoCo and DiLoCo at $K \neq 1$ in these experiments as their batch sizes were not sufficiently swept to determine B_{crit} at each scale. The power laws are fit using L-BFGS as described in the previous section. Figure 13 (a) plots the power law fit for Muon DP, AdamW DP, MuLoCo K=1, and DiLoCo K=1. We observe that Muon DP and MuLoCo have much larger scaling exponents than DiLoCo and AdamW DP. This demonstrates that their critical batch sizes will grow with scale at a faster rate. We also find that MuLoCo and DiLoCo both have larger prefactors than their data-parallel baselines, meaning they will have a slightly larger critical batch size at each scale.

Finding 10: MuLoCo K=1 is projected to have larger critical batch sizes than DP Muon at scale.

Training time efficiency at iso-loss. The largest-scale pre-training runs often take over a month of training time to complete. They are primarily bottlenecked by the number of sequential gradient descent steps required



(a) Iso-loss training-time efficiency

(b) Critical batch-size scaling

Figure 13 MuLoCo achieves substantial training-time gains at iso-loss due to its large critical batch sizes. (left) Reports the iso-loss training-time efficiency of different optimizers relative to DP AdamW. The ratio $T_{\text{AdamW}}(L)/T_{\text{opt}}(L)$ measures how many more sequential FLOPs AdamW requires to reach the same loss L, assuming sufficient parallel and well-interconnected accelerators to realize the speedup from larger critical batch sizes. At observed scales, MuLoCo reaches the same loss as AdamW in roughly $\sim$4.5$\times$ fewer sequential FLOPs. (right) Power-law fits of the critical batch size (CBS) versus training tokens show that MuLoCo's CBS is larger than Muon data-parallel (DP) and significantly larger than AdamW DP and DiLoCo, indicating a wider regime of near-lossless batch scaling. *Dotted curves indicate extrapolation beyond the range of available data.*

for strong performance, which is directly determined by batch size used. While the strongest possible final performance is achieved at the optimal batch size (B_{opt}), to maintain reasonable training time costs for the largest runs, practitioners generally train at the critical batch size (B_{crit}), since with enough training data $B_{\text{crit}} >> B_{\text{opt}}$. Therefore, optimizers should not only be compared w.r.t their best possible performance but also based on their training time to loss at B_{crit}. A reasonable proxy for training time is $T \propto C/B_{\text{crit}}(C)$, the amount of FLOPs that must be computed sequentially when training at B_{crit} (Bergsma et al., 2025). In Figure 13 (a), we directly compare the iso-loss training-time efficiency of different optimizers relative to DP AdamW, finding that K=1 MuLoCo has the strongest training-time advantage.

To accomplish this, for each optimizer m we fit two scaling laws from our training runs: a loss curve $L_m(C) = a_m C^{b_m} + L_0$ with a shared irreducible loss L_0 across all K=1 methods, and a critical batch size scaling law $B_{\text{crit},m}(D) = A_m D^{\alpha_m}$, where D denotes training tokens and we use the Chinchilla-optimal relationship $D = 20N$, $C = 6ND$. Given a target loss L, we invert the loss scaling law to obtain the compute $C_m(L)$ required by each optimizer, and then evaluate the corresponding training-time proxy $T_m(L) = C_m(L)/B_{\text{crit},m}(C_m(L))$. The iso-loss training-time efficiency is then defined as $T_{\text{AdamW}}(L)/T_{\text{opt}}(L)$: a value greater than one indicates that optimizer m reaches the same loss as AdamW in fewer sequential FLOPs.

This metric captures the practical advantage of optimizers with larger critical batch sizes: they can absorb more data-parallel workers without diminishing returns, training faster in wall-clock time for a fixed target loss. Equivalently, the iso-loss training-time ratio decomposes into a compute-efficiency term and a parallelism term:

$$\frac{T_{\text{AdamW}}(L)}{T_{\text{opt}}(L)} = \underbrace{\frac{C_{\text{AdamW}}(L)}{C_{\text{opt}}(L)}}_{\text{compute savings}} \times \underbrace{\frac{B_{\text{crit}}^{(m)}\big(C_m(L)\big)}{B_{\text{crit}}^{(\text{AdamW})}\big(C_{\text{AdamW}}(L)\big)}}_{\text{parallelism advantage}} . \tag{6}$$

We find that K=1 MuLoCo reaches the same loss as DP AdamW in roughly $\sim$4.5$\times$ fewer sequential steps at observed scales. This advantage is much larger than its compute-only savings because MuLoCo combines improved compute scaling with a substantially larger critical batch size.

Finding 11: At iso-loss, K=1 MuLoCo reaches the same loss as DP AdamW in roughly ∼5× fewer sequential steps at observed scales.

Pareto optimal training-time performance tradeoff. The iso-loss analysis above shows that MuLoCo can reach a target loss in substantially fewer sequential FLOPs than DP AdamW by combining improved compute efficiency with a larger critical batch size. However, this comparison is not FLOP-matched: because MuLoCo is itself more FLOP-efficient, it generally reaches lower loss than AdamW at the same compute budget before any training-time advantage from larger batch sizes is considered. We therefore also examine the FLOP-matched setting in Figure 1 (b), which compares final evaluation loss as a function of FLOPs per batch size, a proxy for sequential training time. At the 3.1B scale, MuLoCo K=1 lies on the Pareto frontier: it achieves stronger final performance than AdamW DP, Muon DP, and DiLoCo K=1, while also supporting the largest critical batch size. Notably, at its critical batch size, MuLoCo matches the performance of DP Muon at its optimal batch size and outperforms AdamW and DiLoCo at all batch sizes. Thus, among the optimizers compared, MuLoCo K=1 provides the best training-time-performance tradeoff in the FLOP-matched regime.

Finding 12: In a FLOP-matched comparison, K=1 MuLoCo lies on the Pareto frontier between final performance and training time, achieving the strongest loss while supporting the largest critical batch size.

7.3 Scaling up to 15B

In this section, we use the datapoints from our scaling study to extrapolate optimal hyperparameters to 15B scale, train with these hyperparameters, and evaluate the performance of the resulting models. While reporting the optimal performance is not guaranteed for each method since we did not sweep at this scale, these experiments provide us with an understanding of the performance attainable from different optimizers, an understanding of how large batch training and communication overhead contributed to wall-clock training time, and an increased confidence in the scalability of our method.

Extrapolating optimal hyperparameters. Using the data collected in our scaling experiments, we fit three new power laws: (1) $\lambda(N) = aN^{\alpha}$ (weight decay), (2) $\eta(N) = aN^{\alpha}$ (learning rate), and (3) $B_{\text{opt}}(D) = aD^{\alpha}$ (optimal batch size). Following our tuning protocol from section 5, the weight decay power law is fit to the optimal value found at $B = 1$M and extrapolated in this regime. The optimal value found is subsequently rescaled according to the extrapolated optimal batch size. For learning rate and batch size we simply fit power laws to their optimal values for each optimizer. It should be noted that our power laws for with B_{opt} were fit with K=1 MuLoCo, having $B_{\text{opt}} = 4$M at 3.1B scale. However, further sweeping later revealed that this was a local minimum (see Fig. 12). This means that K=1 MuLoCo's batch size used at 15B scale is likely closer to the B_{crit} than B_{opt} and that K=1 MuLoCo's optimal final performance is likely slightly under-reported. Further details in addition to the optimal values used are reported in E.2 of the appendix.

Table 3 Zero-shot LM Evaluation Results for K=1 and K=16 optimizers at 15B scale. We extrapolate optimal hyperparameters for K=1 and K=16 optimizers and train 15B parameter models with them. We report accuracy for MMLU, WinoGrande, and normalized accuracy for HellaSwag, PIQA, ARC-E, ARC-C, OBQA.

Model Size	Optimizer	Eval. Loss	MMLU	HellaSwag	WinoGrande	PIQA	ARC-E	ARC-C	OBQA	Mean Acc.
15B	DP AdamW	1.8873	57.20%	77.69%	71.19%	81.39%	78.11%	52.30%	45.40%	66.18%
15B	DP Muon	**1.8642**	59.14%	79.05%	71.19%	81.12%	77.23%	53.07%	46.80%	**66.80**%
15B	DiLoCo K=1	1.8914	59.80%	79.06%	71.43%	81.34%	77.99%	52.56%	46.40%	66.94%
15B	MuLoCo K=1	**1.8839**	59.49%	79.06%	71.11%	81.61%	78.11%	52.73%	46.60%	**66.96**%
15B	DiLoCo K=16	**1.9062**	54.15%	77.59%	69.30%	81.72%	77.65%	52.56%	46.80%	65.68%
15B	MuLoCo K=16	1.9168	56.96%	77.18%	70.72%	80.74%	76.60%	53.16%	46.20%	**65.94**%

Evaluating final performance. Table 3 reports the final loss values obtained for models trained with different optimizers in addition to their zero-shot performance across MMLU, HellaSwag, WinoGrande, PIQA, ARC-E, ARC-C, and OBQA. Across optimizers, we find that all methods converge to *nearly* the same average downstream accuracy, with MuLoCo (K=1), DiLoCo (K=1), and data-parallel Muon performing marginally stronger, while the K=16 variants and DP AdamW perform slightly worse. Importantly, as shown in the per-step view of Figure 19, MuLoCo (K=1) trains at the largest batch size (17M) while reaching nearly identical final evaluation loss and evaluation performance to DiLoCo (K=1) and DP Muon both trained for

many more steps. This demonstrates that MuLoCo $K = 1$ can leverage more parallelism without sacrificing performance. A similar advantage holds for MuLoCo (K=16), which reached comparable performance to DiLoCo (K=16) and DP AdamW while also having a larger batch size than both.

Evaluating wall-clock training time when accounting for batch size, optimizer step times, and communication overhead. We also collected system-level metrics during training, enabling wall-clock analyses under networking constraints (Figure 20). In bandwidth-constrained regimes (10 Gbit/s), the communication-efficient optimizers dominate: both K=16 methods train substantially faster than K=1 MuLoCo, despite their smaller batch sizes, with MuLoCo (K=16) emerging as the fastest configuration due to its larger extrapolated optimal batch size. As bandwidth increases (800 to 12,800 Gbit/s), communication becomes less of a bottleneck and leveraging more compute in parallel becomes more important. In the high bandwidth regime, the ability of MuLoCo (K=1) to leverage very large batch sizes becomes advantageous, leading it ultimately train the fastest.

Finding 13: When training under bandwidth constraints, K=16 MuLoCo has the fastest wall-clock training time, while K=1 MuLoCo is fastest in high-bandwidth settings.

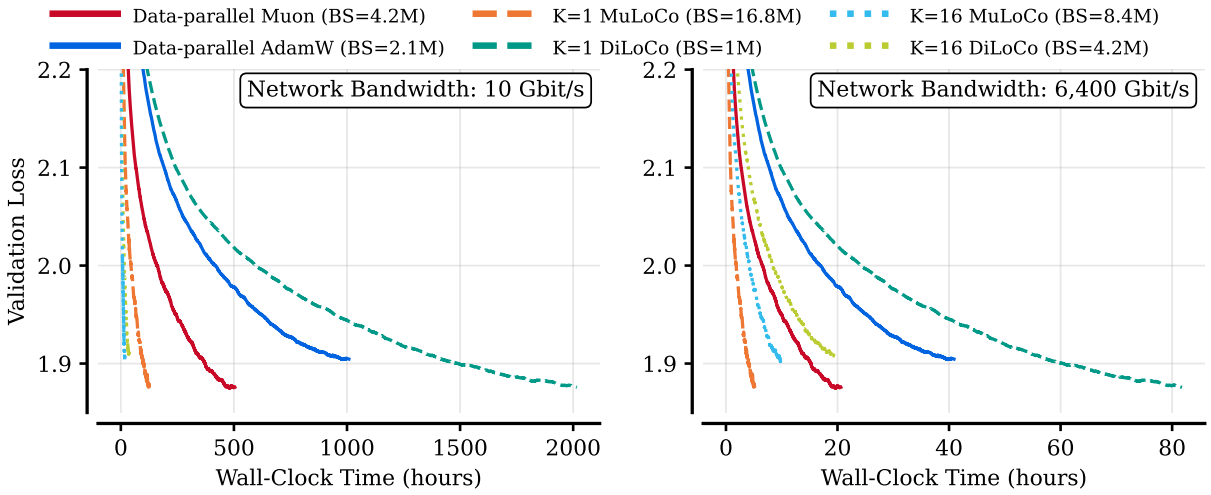


Figure 14 Idealized wall-clock training time for 15B runs under low- and high-bandwidth networking. We estimate training time by combining (i) network communication time, (ii) optimizer step time, and (iii) forward/backward (FW/BW) compute time. The left panel reports validation loss versus wall-clock time assuming a low-bandwidth network, while the right panel reports the same quantities in a datacenter setting with high bandwidth. In the low-bandwidth regime, **K=16 MuLoCo** achieves the fastest time-to-loss and dominates the DiLoCo baselines; in the datacenter regime, **K=1 MuLoCo** achieves the fastest time-to-loss. In both cases, MuLoCo's larger batch sizes translate into faster wall-clock training to comparable loss than corresponding DiLoCo configurations.

8 Limitations

We have made our best effort to conduct a sound empirical study. However, as with most empirical works, we could not exhaustively run all relevant experiments, leading to natural limitations from settings that were not tried. Listed in no particular order, we acknowledge that out study has the following limitations. Firstly, our exploration of MuLoCo's performance at scale is limited to experiments training 150M-15B dense transformers in a 20TPP setting. Therefore, the scaling trends we predict may not hold for models that are several orders of magnitude larger. Secondly, our study is limited to studying dense transformers with no exploration of other popular foundation model architectures (e.g., MoEs, SSMs, or Hybrid architectures). Thirdly, we do not explore outer optimizers beyond Nesterov SGD. Finally, we do not prove convergence of the method.

9 Conclusion

In conclusion, we have compared the performance of DiLoCo to MuLoCo across a range of workloads relevant to distributed optimization and as training is scaled up. We established that MuLoCo outperforms DiLoCo under long communication intervals, compressed communication, when using streaming updates, and even in wall-clock training time. Beyond improving over DiLoCo in absolute terms, we even found that MuLoCo's performance relative to DP Muon scales better than DiLoCo's performance relative to DP AdamW as the number of workers increases. Furthermore, we found that MuLoCo's critical batch size is larger than DiLoCo's at all worker counts, leading to faster wall-clock training times and possibly warranting training across datacenters to take advantage of the additional data parallelism. Finally, we found that K=1 MuLoCo has a Pareto-optimal performance-training-time tradeoff among all optimizers in our study. Overall, our results demonstrate that MuLoCo is a flexible distributed optimization algorithm. With its large critical batch sizes, stronger performance than DP Muon at K=1, and its improved worker scaling over DiLoCo, MuLoCo has significant advantages for large-scale training in both low-bandwidth and high-bandwidth settings.

Acknowledgements

We are particularly grateful to Lin Xiao for providing feedback on our manuscript and guidance on our experiments throughout. We thank Jack Rae and Tom Gunter for providing practical guidance regarding our scaling law and critical batch size experiments. We thank Andrei Mircea and Ekaterina Lobacheva for insightful discussions about our pseudogradient analysis experiments. We thank Howard Huang and Tushar Jain for their advice in resolving TorchTitan and TorchFT bugs. We would also like to thank Zeyuan Allen-Zhu, Konstantin Mishchenko, Ashok Cutkosky, Ziqui Bu, Alice Yang, Lisa jin, Andrey Gromov, Maissam Barkeshli, and Sanae Lotfi for fruitful comments and research discussions.

We acknowledge support FRQNT New Scholar [*E.B.*], the FRQNT Doctoral (B2X) scholarship [*B.T.*], the Canada CIFAR AI Chair Program [*I.R.*], and the Canada Excellence Research Chairs Program in Autonomous AI [*I.R.*].

References

Istabrak Abbes, Gopeshh Subbaraj, Matthew Riemer, Nizar Islah, Benjamin Thérien, Tsuguchika Tabaru, Hiroaki Kingetsu, Sarath Chandar, and Irina Rish. Revisiting replay and gradient alignment for continual pre-training of large language models. *CoRR*, abs/2508.01908, 2025. https://doi.org/10.48550/arXiv.2508.01908.

Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates, 2025. https://arxiv.org/abs/2504.05295.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification, and local computations, 2019.

Shane Bergsma, Nolan Simran Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in LLM pre-training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. https://openreview.net/forum?id=bFXbLQzRoZ.

Tamay Besiroglu, Ege Erdil, Matthew Barnett, and Josh You. Chinchilla scaling: A replication attempt. *arXiv preprint arXiv:2404.10102*, 2024. https://arxiv.org/abs/2404.10102.

Åke Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971. Cited on pages 1 and 8.

Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco, March 2025. https://arxiv.org/abs/2503.09799.

Ziheng Cheng and Margalit Glasgow. Convergence of distributed adaptive optimization with local updates. In *The Thirteenth International Conference on Learning Representations*, 2025. https://openreview.net/forum?id=VNg7srnvD9.

DeepSeek-AI. Deepseek-v3 technical report, 2024. https://arxiv.org/abs/2412.19437.

Aaron Defazio, Konstantin Mishchenko, Parameswaran Raman, Hao-Jun Michael Shi, and Lin Xiao. Smoothing diloco with primal averaging for faster training of llms. *CoRR*, abs/2512.17131, 2025. https://doi.org/10.48550/arXiv.2512.17131.

Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023. https://arxiv.org/abs/2304.03208.

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming diloco with overlapping communication: Towards a distributed free lunch, January 2025. https://arxiv.org/abs/2501.18512.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. http://arxiv.org/abs/1706.02677.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. doi: 10.52202/079017-2427.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. https://doi.org/10.48550/arXiv.2203.15556.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. https://doi.org/10.1214/aoms/1177703732.

Alex Iacob, Andrej Jovanovic, Mher Safaryan, Meghdad Kurmanji, Lorenzo Sani, Samuel Horváth, William F. Shen, Xinchi Qiu, and Nicholas D. Lane. MT-DAO: Multi-timescale distributed adaptive optimizers with local updates. In *The Fourteenth International Conference on Learning Representations*, 2026a. https://openreview.net/forum?id=5yPP238v4c.

Alex Iacob, Lorenzo Sani, Mher Safaryan, Paris Giampouras, Samuel Horváth, Andrej Jovanovic, Meghdad Kurmanji, Preslav Aleksandrov, William F. Shen, Xinchi Qiu, and Nicholas D. Lane. DES-LOC: Desynced low communication adaptive optimizers for foundation models. In *The Fourteenth International Conference on Learning Representations*, 2026b. https://openreview.net/forum?id=6N2qFixxYZ.

Sami Jaghouar, Jack Min Ong, Manveer Basra, Fares Obeid, Jannik Straube, Michael Keiblinger, Elie Bakouch, Lucas Atkins, Maziyar Panahi, Charles Goddard, Max Ryabinin, and Johannes Hagemann. INTELLECT-1 technical report. *CoRR*, abs/2412.01152, 2024a. https://doi.org/10.48550/arXiv.2412.01152.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. Opendiloco: An open-source framework for globally distributed low-communication training, July 2024b. https://arxiv.org/abs/2407.07852.

Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J. Storkey. Three factors influencing minima in SGD. *CoRR*, abs/1711.04623, 2017. http://arxiv.org/abs/1711.04623.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. https://web.archive.org/web/20250122060345/https://kellerjordan.github.io/posts/muon/.

Charles-Étienne Joseph, Benjamin Thérien, Abhinav Moudgil, Boris Knyazev, and Eugene Belilovsky. Meta-learning optimizers for communication-efficient learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. https://openreview.net/forum?id=uRbf9ANAns.

Andrej Jovanović, Alex Iacob, Mher Safaryan, Ionut-Vlad Modoranu, Lorenzo Sani, William F. Shen, Xinchi Qiu, Dan Alistarh, and Nicholas D. Lane. Lordo: Distributed low-rank optimization with infrequent communication, 2026. https://arxiv.org/abs/2602.04396.

Dominik Kallusky, Vinay Rao, Vishal Nandavanam, and Hao-Jun Michael Shi. SNOO: step-k nesterov outer optimizer - the surprising effectiveness of nesterov momentum applied to pseudo-gradients. *CoRR*, abs/2510.15830, 2025. https://doi.org/10.48550/arXiv.2510.15830.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Róbert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucinska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju-yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle K. Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clément Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry (Dima) Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report. *CoRR*, abs/2503.19786, 2025. https://doi.org/10.48550/arXiv.2503.19786.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3252–3261, 2019. https://arxiv.org/abs/1901.09847.

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 *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. https://openreview.net/forum?id=H1oyRlYgg.

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 1970. Cited on pages 1 and 8.

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 pretraining. In *The Thirteenth International Conference on Learning Representations*, 2025. https://openreview.net/forum?id=SFN6Wm7YBI.

Tao Lin, Sebastian U. Stich, and Martin Jaggi. Don't use large mini-batches, use local SGD. *CoRR*, abs/1808.07217, 2018.

Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Extrapolation for large-batch training in deep learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 6059–6069. PMLR, 2020.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training, February 2025. https://arxiv.org/abs/2502.16982.

Llama-Team. The llama 3 herd of models, 2024. https://arxiv.org/abs/2407.21783.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018. http://arxiv.org/abs/1812.06162.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

William Merrill, Shane Arora, Dirk Groeneveld, and Hannaneh Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. *CoRR*, abs/2505.23971, 2025. https://doi.org/10.48550/arXiv.2505.23971.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. http://arxiv.org/abs/1803.02999.

Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980. ISSN 00255718, 10886842. http://www.jstor.org/stable/2006193.

NVIDIA. *NVIDIA DGX SuperPOD: Next-Generation Scalable Infrastructure for AI Leadership Reference Architecture*. NVIDIA Corporation, 2024. Reference Architecture RA-11333-001 V11.

Bowen Peng, Jeffrey Quesnelle, and Diederik P Kingma. Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021. https://arxiv.org/abs/2003.00295.

Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8253–8265. PMLR, 2020. http://proceedings.mlr.press/v119/rothchild20a.html.

Lorenzo Sani, Alex Iacob, Zeyu Cao, Bill Marino, Yan Gao, Tomas Paulik, Wanru Zhao, William F. Shen, Preslav Aleksandrov, Xinchi Qiu, and Nicholas D. Lane. The future of large language model pre-training is federated. *arXiv preprint arXiv:2405.10853*, 2024. https://arxiv.org/abs/2405.10853.

Amir Sarfi, Benjamin Thérien, Joel Lidin, and Eugene Belilovsky. Communication efficient LLM pre-training with sparseloco. *CoRR*, abs/2508.15706, 2025. https://doi.org/10.48550/arXiv.2508.15706.

Rylan Schaeffer, Noam Levi, Andreas Kirsch, Theo Guenais, Brando Miranda, Elyas Obbad, and Sanmi Koyejo. Evaluating the robustness of chinchilla compute-optimal scaling. *ICLR 2026 Conference Submission (OpenReview) and arXiv preprint arXiv:2509.23963*, 2025. https://arxiv.org/pdf/2509.23963.

Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J. Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of muon for pretraining, 2025a. https://arxiv.org/abs/2505.02222.

Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J. Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of muon for pretraining. *CoRR*, abs/2505.02222, 2025b. https://doi.org/10.48550/arXiv.2505.02222.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *CoRR*, abs/1911.08772, 2019.

Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. *CoRR*, abs/1809.07599, 2018.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2024. https://arxiv.org/abs/2412.02595.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. https://arxiv.org/abs/2507.20534.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1905.13727.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. SOAP: Improving and stabilizing shampoo using adam for language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. https://openreview.net/forum?id=IDxZhXrpNf.

Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. Zero++: Extremely efficient collective communication for giant model training. *CoRR*, abs/2306.10209, 2023a. https://doi.org/10.48550/arXiv.2306.10209.

Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael G. Rabbat. Slowmo: Improving communication-efficient distributed SGD with slow momentum. *CoRR*, abs/1910.00643, 2019.

Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. CocktailSGD: Fine-tuning foundation models over 500Mbps networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36058–36076. PMLR, 23–29 Jul 2023b.

Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. *CoRR*, abs/2405.13698, 2024. https://doi.org/10.48550/arXiv.2405.13698.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng

Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E. Dahl, Christopher J. Shallue, and Roger B. Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8194–8205, 2019a. https://proceedings.neurips.cc/paper/2019/hash/e0eacd983971634327ae1819ea8b6214-Abstract.html.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean P. Foster, and Sham M. Kakade. How does critical batch size scale in pre-training? In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b. https://proceedings.neurips.cc/paper_files/paper/2019/file/90fd4f88f588ae64038134f1eeaa023f-Paper.pdf.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. https://arxiv.org/abs/2403.03507.

Liuzhi Zhou, Yu He, Kun Zhai, Xiang Liu, Sen Liu, Xingjun Ma, Guangnan Ye, Yu-Gang Jiang, and Hongfeng Chai. Fedcada: Adaptive client-side optimization for accelerated and stable federated learning. *arXiv preprint arXiv:2405.11811*, 2024. https://arxiv.org/abs/2405.11811.

Appendix

Contents

1 Introduction **1**

2 Background **3**

3 Related Work **5**
- 3.1 Distributed optimization with local steps . . . 5
- 3.2 Distributed optimizers with compressed communication . . . 5
- 3.3 Single worker DiLoCo . . . 5
- 3.4 Large-Batch Training and Critical Batch Size . . . 6

4 Method **6**
- 4.1 MuLoCo . . . 6
- 4.2 Analyzing the benefits of Muon pseudogradients over AdamW . . . 7
- 4.3 Relating Muon and AdamW Pseudogradient Spectra to Per-step Alignment . . . 8

5 Empirical Evaluation **11**

6 Results—Communication Efficient Ablations & Wall-Clock Time **12**
- 6.1 MuLoCo vs DiLoCo: Increasing K . . . 13
- 6.2 MuLoCo vs DiLoCo: Increasing H . . . 14
- 6.3 MuLoCo vs DiLoCo: Pseudogradient Compression . . . 14
- 6.4 Streaming MuLoCo . . . 16
- 6.5 Wall-clock training time . . . 16

7 Results—Scaling up **17**
- 7.1 Extrapolating MuLoCo's performance when scaling compute. . . . 17
- 7.2 On MuLoCo's large critical batch size . . . 20
- 7.3 Scaling up to 15B . . . 22

8 Limitations **23**

9 Conclusion **24**

A MuLoCo Algorithm Block **31**

B Proof of Proposition 4.2 **31**

C Extended Results **33**
- C.1 MuLoCo vs DiLoCo: Pseudogradient Compression . . . 33
- C.2 Extended Scaling Law Experiments . . . 36
- C.3 Extended 15B results . . . 39

D Extended pseudogradient analysis **42**

E Optimal hyperparameters **42**
- E.1 Hyperparameter Sweep Details . . . 42
- E.2 Extropolating HPs to 15B scale . . . 46

F A Robuts Evaluation Loss Estimate **48**

A MuLoCo Algorithm Block

We present non-compressed and compressed versions of MuLoCo in Algorithms 1 and 2, respectively. The compressed version of the algorithm is used in the compression experiments of section 6.3, while Algorithm 1 is used elsewhere.

Algorithm 1 MuLoCo (No Compression).

Input: N (communication steps), K (workers), H (local steps), t (local step counter), data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$, initial model $\theta^{(0)}$, outer momentum μ, outer LR η_{out}
$u^{(0)} \leftarrow 0; \quad \theta_i^{(1)} \leftarrow \theta^{(0)} \quad \forall i; \; t \leftarrow 0$
for $n = 1 \ldots N$ **do**
  for $i = 1 \ldots K$ **in parallel** **do**
    for $h = 1 \ldots H$ **do**
      $t \leftarrow t + 1; \quad (x, y) \sim \mathcal{D}_i; \quad \mathcal{L} \leftarrow f_\theta(x, y; \theta_i^{(t)})$
      $\theta_i^{(t)} \leftarrow \text{Muon}(\theta_i^{(t)}, \nabla\mathcal{L})$
    end for
    $\Delta_i^{(t)} \leftarrow \theta^{(t-H)} - \theta_i^{(t)}$ *(parameter delta)*
  end for
  $\Delta^{(t)} \leftarrow \frac{1}{K}\sum_{i=1}^{K} \Delta_i^{(t)}$ *(all-reduce outer delta)*
  $u^{(t)} \leftarrow \mu u^{(t-H)} + \eta_{\text{out}}\Delta^{(t)}$ *(Outer Momentum)*
  $\theta^{(t)} \leftarrow \theta^{(t-1)} - \mu u^{(t)} - \eta_{\text{out}}\Delta^{(t)}$ *(Outer Update)*
end for

Algorithm 2 MuLoCo with compressed communication and optional EF.

Input: N (communication steps), K (workers), H (local steps), t (local step counter), data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$, initial model $\theta^{(0)}$, compressor $\mathcal{C}$, outer momentum μ, outer LR η_{out}, error feedback flag EF
$u^{(0)} \leftarrow 0; \quad \theta_i^{(1)} \leftarrow \theta^{(0)} \quad \forall i; \; t \leftarrow 0$
if EF then
  $\mathcal{E}_i^{(0)} \leftarrow 0 \quad \forall i$
end if
for $n = 1 \ldots N$ **do**
  for $i = 1 \ldots K$ **in parallel** **do**
    for $h = 1 \ldots H$ **do**
      $t \leftarrow t + 1; \quad (x, y) \sim \mathcal{D}_i; \quad \mathcal{L} \leftarrow f_\theta(x, y; \theta_i^{(t)})$
      $\theta_i^{(t)} \leftarrow \text{Muon}(\theta_i^{(t)}, \nabla\mathcal{L})$
    end for
    $\Delta_i^{(t)} \leftarrow \theta^{(t-H)} - \theta_i^{(t)}$ *(parameter delta)*
    if EF then
      $\mathcal{E}_i^{(t)} \leftarrow \beta\,\mathcal{E}_i^{(t-H)} + \Delta_i^{(t)}$
      $\Delta_i^{(t)} \leftarrow \mathcal{C}(\mathcal{E}_i^{(t)})$ *(compress EF)*
      $\mathcal{E}_i^{(t+1)} \leftarrow \mathcal{E}_i^{(t)} - \Delta_i^{(t)}$
    else
      $\Delta_i^{(t)} \leftarrow \mathcal{C}(\Delta_i^{(t)})$ *(compress parameter delta)*
    end if
  end for
  $\Delta^{(t)} \leftarrow \frac{1}{K}\sum_{i=1}^{K} \Delta_i^{(t)}$ *(all-reduce outer delta)*
  $u^{(t)} \leftarrow \mu u^{(t-H)} + \eta_{\text{out}}\Delta^{(t)}$ *(Outer Momentum)*
  $\theta^{(t)} \leftarrow \theta^{(t-1)} - \mu u^{(t)} - \eta_{\text{out}}\Delta^{(t)}$ *(Outer Update)*
end for

B Proof of Proposition 4.2

Notation. For a matrix $A \in \mathbb{R}^{m\times n}$, we denote by $\sigma_1(A) \geq \cdots \geq \sigma_r(A) \geq 0$ its singular values, with $r := \min\{m, n\}$. The Frobenius inner product is $\langle A, B\rangle_F := \text{Tr}(A^\top B)$, and the Frobenius norm is $\|A\|_F := \sqrt{\langle A, A\rangle_F}$. The nuclear norm is $\|A\|_* := \sum_{j=1}^{r} \sigma_j(A)$.

Proposition B.1 (Pseudogradient nuclear norm depends on optimizer step alignment)**.** *Fix $m, n \in \mathbb{N}$ and let $r := \min\{m, n\}$. Consider a pseudogradient of the form*

$$\Psi \;=\; \frac{1}{K}\sum_{k=1}^{K}\sum_{h=1}^{H} \alpha_{t+h}\,\psi^{(t+h,k)} \in \mathbb{R}^{m\times n}, \qquad \alpha_{t+h} \geq 0.$$

Let $\Psi = U\Sigma V^\top$ be its singular value decomposition and define the orthonormal factor $\Psi^\star := UV^\top$. Let the cosine similarity between a given optimizer step and $\Psi^\star$ be

$$\rho^{(t+h,k)} \;:=\; \frac{\langle \psi^{(t+h,k)}, \Psi^\star\rangle_F}{\|\psi^{(t+h,k)}\|_F\;\|\Psi^\star\|_F}.$$

Then the nuclear norm of Ψ satisfies

$$\|\Psi\|_* = \frac{\sqrt{r}}{K}\sum_{k=1}^{K}\sum_{h=1}^{H} \rho^{(t+h,k)}\alpha_{t+h}\,\|\psi^{(t+h,k)}\|_F.$$

Proof. Let $\Psi = U\Sigma V^\top$ be the singular value decomposition of Ψ, and recall the definition of its *orthonormal factor* $\Psi^\star := UV^\top$.

We first show that $\langle \Psi, \Psi^\star \rangle_F = \|\Psi\|_*$. Indeed,

$$\begin{aligned}\langle \Psi, \Psi^\star \rangle_F &= \mathrm{Tr}\big(\Psi^\top \Psi^\star\big) = \mathrm{Tr}\big((U\Sigma V^\top)^\top (UV^\top)\big) \\ &= \mathrm{Tr}\big((V\Sigma U^\top)(UV^\top)\big) = \mathrm{Tr}\big(V\Sigma (U^\top U) V^\top\big) \\ &= \mathrm{Tr}\big(V\Sigma V^\top\big) = \mathrm{Tr}(\Sigma) = \sum_{j=1}^{r} \sigma_j(\Psi) = \|\Psi\|_*. \end{aligned} \tag{7}$$

By linearity of the Frobenius inner product, we can expand

$$\begin{aligned}\langle \Psi, \Psi^\star \rangle_F &= \left\langle \frac{1}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \alpha_{t+h} \psi^{(t+h,k)},\ \Psi^\star \right\rangle_F \\ &= \frac{1}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \alpha_{t+h} \, \langle \psi^{(t+h,k)}, \Psi^\star \rangle_F. \end{aligned} \tag{8}$$

Substituting the definition of cosine similarity,

$$\rho^{(t+h,k)} \ := \ \frac{\langle \psi^{(t+h,k)}, \Psi^\star \rangle_F}{\|\psi^{(t+h,k)}\|_F \ \|\Psi^\star\|_F},$$

we obtain

$$\langle \Psi, \Psi^\star \rangle_F = \frac{1}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h} \|\psi^{(t+h,k)}\|_F \|\Psi^\star\|_F. \tag{9}$$

Finally, since $\Psi^\star = UV^\top$ has singular values equal to 1 for $j \le r$, we have

$$\|\Psi^\star\|_F = \sqrt{\sum_{j=1}^{r} \sigma_j(\Psi^\star)^2} = \sqrt{r}.$$

Combining this with equation 7, equation 9 yields

$$\|\Psi\|_* = \langle \Psi, \Psi^\star \rangle_F = \frac{\sqrt{r}}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h} \, \|\psi^{(t+h,k)}\|_F,$$

□

Corollary B.2 (Muon pseudogradient)**.** *In the setting of Proposition 4.2, assume that the inner-step optimizer updates, $\psi^{(t+h,k)}$, are produced by Newton Shulz Iteration, making them orthonormal; then*

$$\|\Psi\|_* = \frac{r}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h}.$$

Proof. This follows from proposition 4.2 since $\psi^{(t+h,k)}$ is orthonormal we have $\|\psi^{(t+h,k)}\|_F = \sqrt{r}$. □

Corollary B.3 (AdamW pseudogradient)**.** *In the setting of Proposition 4.2, for general inner-step matrices $\psi^{(t+h,k)}$ (e.g., AdamW effective steps), one always has*

$$\|\Psi\|_* = \frac{\sqrt{r}}{K} \sum_{k=1}^{K} \sum_{h=1}^{H} \rho^{(t+h,k)} \alpha_{t+h} \, \|\psi^{(t+h,k)}\|_F.$$

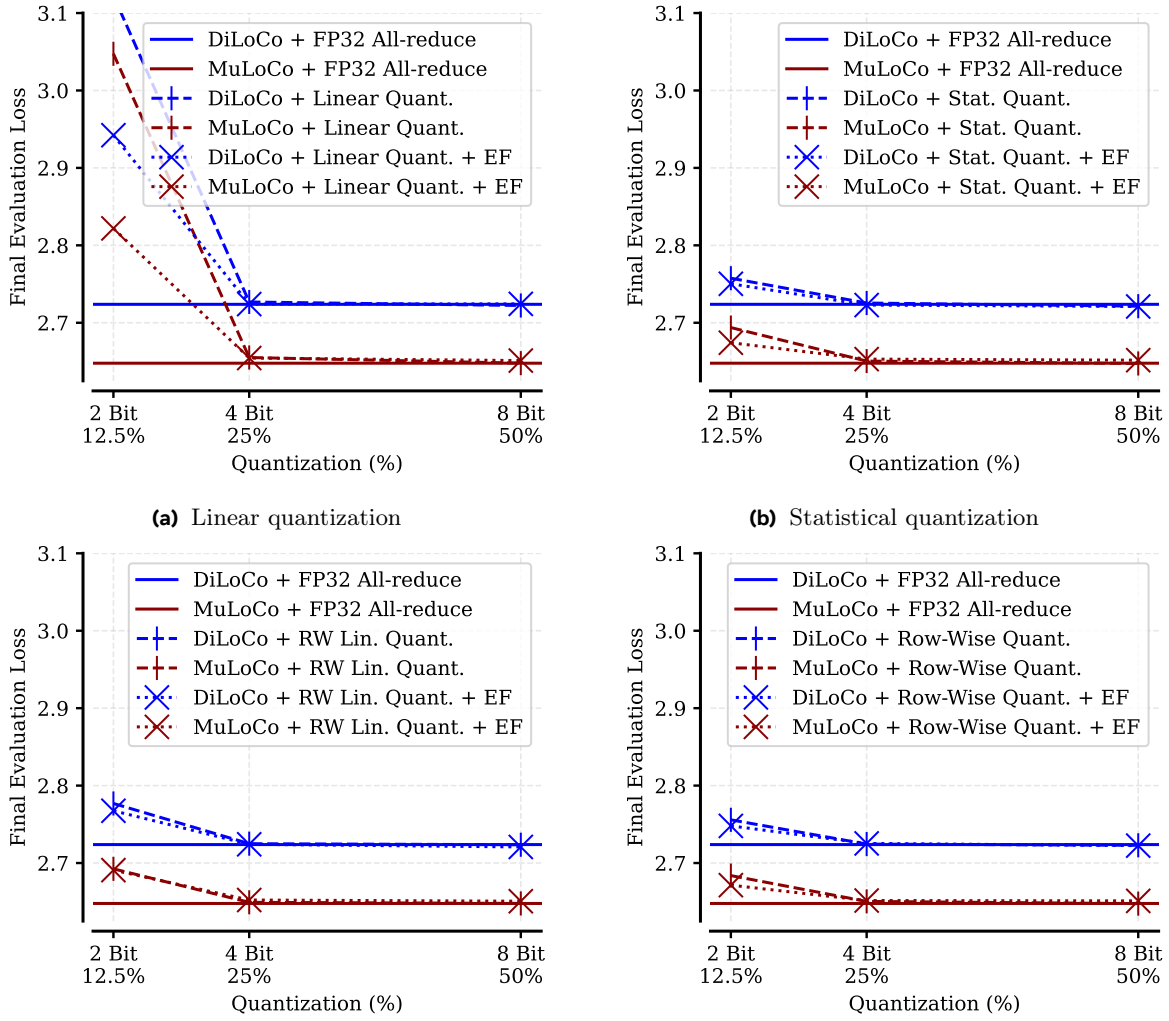
Proof. This is a restatement of proposition 4.2 to illustrate that it applies to AdamW pseudogradients. □

C Extended Results

This section reviews complementary and additional results to the main manuscript: (C.1) additional experiments regarding compressed communication, (C.2) additional scaling law results, and (C.3) additional results detailing our 15B training runs.

C.1 MuLoCo vs DiLoCo: Pseudogradient Compression

Figure 15 reports the performance for all quantization methods, including row-wise versions of statistical and linear quantization. We observe that across all settings, 4-bit quantization is effectively lossless and 2-bit quantization remains strong, *except* for global linear quantization, which shows a clear degradation at 2 bits. In all cases, MuLoCo outperforms DiLoCo, and enabling EF provides little additional benefit. In figure 16 we observe that even quantizing to 4-bits can have a substantial advantage when communication is the bottleneck. As demonstrated in our experiments, this comes at no cost to performance.



(a) Linear quantization

(b) Statistical quantization

(c) Row-wise linear quantization

(d) Row-wise statistical quantization

Figure 15 Quantization ablations. We compare MuLoCo and DiLoCo under 8-bit (50%), 4-bit (25%), and 2-bit (12.5%) quantized communication, with and without error feedback (EF). Across all settings, 4-bit quantization is effectively lossless and 2-bit quantization remains strong, *except* for global linear quantization, which shows a clear degradation at 2 bits. In all cases, MuLoCo outperforms DiLoCo, and enabling EF provides little additional benefit.

Assumptions around the collective used. For our quantization experiments, we assume that collective communi-

cation is implemented using a bandwidth-efficient all-to-all reduce-scatter and subsequent ring all-gather (Wang et al., 2023a). This choice is made instead of (1) using a bandwidth-expensive all-gather of full quantized parameters and (2) using a ring for the reduce-scatter. Compared to (1), the all-to-all allows us to avoid bandwidth growth in the number of workers, while relative to (2) it avoids compounding quantization errors during the ring-reduction, which is also exacerbated as the number of workers is increased. When using an all-to-all collective for the reduce scatter, all quantized tensors can be reduced in high precision on the same device before being re-quantized for the final all-gather. This required two total quantizations: (1) a quantization before the all-to-all, and (2) a quantization after the high-precision reduction before the all-gather. All our quantization experiments use two quantizations. For our top-k experiments, we assume an all-gather across all parameters and, therefore, only sparsify the tensor once immediately before communication.

Compression results in table format For the reader's convenience, we provide our quantization and top-k results in Tables 5 and 4, respectively.

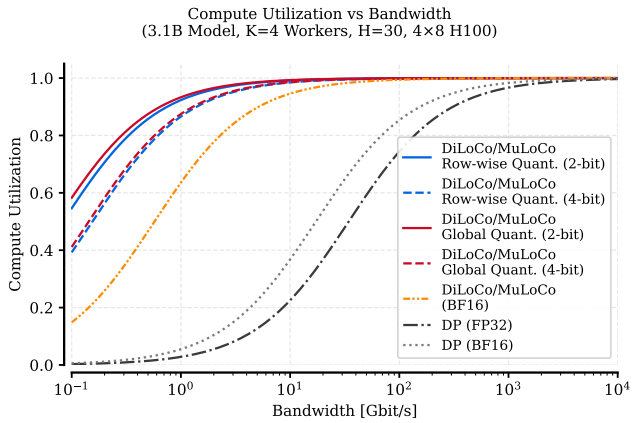


Figure 16 Compute Utilization as a function of network bandwidth when training a 3.1B model with different optimizers and compression schemes. The plot was using timings from Table 9 and assumes that any time not spent communicating is used for computation. We observe that DiLoCo and MuLoCo baselines require two orders of magnitude less bandwidth to achieve 99%+ compute utilization than DP approaches.

Table 4 Final evaluation loss for top-k sparsification.

	DiLoCo		MuLoCo	
Top-K	w/o EF	w/ EF	w/o EF	w/ EF
FP32 (baseline)	2.7238		2.6478	
0.5%	3.3660	3.1646	3.0221	2.9052
1%	3.2231	3.0641	2.9662	2.8633
2.5%	3.0644	2.9402	2.9211	2.7988
5%	2.9630	2.8695	2.8417	2.7580
10%	2.8712	2.8224	2.7777	2.7230
25%	2.7883	2.7529	2.7084	2.6822
50%	2.7437	2.7316	2.6668	2.6566

Table 5 Final evaluation loss for different quantization strategies.

		DiLoCo		MuLoCo	
Compressor	Bits	w/o EF	w/ EF	w/o EF	w/ EF
FP32 (baseline)	-	2.7238		2.6478	
Linear	2-bit	2.9819	2.8695	2.7751	2.7543
	4-bit	2.7268	2.7241	2.6551	2.6544
	8-bit	2.7223	2.7244	2.6478	2.6512
Statistical	2-bit	2.7576	2.7503	2.6938	2.6742
	4-bit	2.7255	2.7229	2.6505	2.6531
	8-bit	2.7214	2.7210	2.6474	2.6517
RW Linear	2-bit	2.7769	2.7675	2.6925	2.6908
	4-bit	2.7250	2.7242	2.6492	2.6522
	8-bit	2.7236	2.7208	2.6478	2.6507
RW Statistical	2-bit	2.7558	2.7480	2.6838	2.6714
	4-bit	2.7245	2.7253	2.6504	2.6516
	8-bit	2.7229	2.7225	2.6476	2.6511

C.2 Extended Scaling Law Experiments

The following section presents further analysis and reports on greater details of our scaling law experiments. Table 6 reports the parameters of scaling laws fit in section 7. We also report the final loss values of all methods in Table 7 and the zero-shot evaluation benchmark scores for 3.1B models in Table 8.

Table 6 Parameters for $L(C) = aC^{\alpha} + L_0$ with joint $L_0 = 1.711$, fitted on all five training scales (150M–3.1B). Residuals are $|\log L_{\text{actual}} - \log L_{\text{pred}}|$.

Method	K	a	α	Train	15B
DP AdamW	1	5677	−0.195	0.003	0.025
DP Muon	1	6584	−0.199	0.002	0.032
DiLoCo	1	6620	−0.199	0.004	0.032
	2	5647	−0.195	0.003	—
	4	5640	−0.194	0.002	—
	8	6015	−0.195	0.002	—
	16	5372	−0.191	0.001	0.040
MuLoCo	1	6927	−0.200	0.002	0.028
	2	6852	−0.200	0.003	—
	4	6851	−0.200	0.002	—
	8	6467	−0.198	0.004	—
	16	6906	−0.199	0.002	0.026
Mean				**0.002**	**0.030**

Table 7 MuLoCo vs DiLoCo scaling study. Percentages show relative change from respective DP baselines. Bold = best (lowest) loss in each column.

	Method	**150M**	**416M**	**914M**	**1.8B**	**3.1B**	**15B** (No Sweep)
DP	Muon	3.124	2.641	2.402	2.246	2.128	**1.875**
	AdamW	3.158	2.682	2.440	2.266	2.145	1.901
K=1	MuLoCo	**3.120** (−0.1%)	**2.638** (−0.1%)	**2.400** (−0.1%)	**2.238** (−0.4%)	**2.122** (−0.3%)	1.878 (+0.2%)
	DiLoCo	3.142 (−0.5%)	2.650 (−1.2%)	2.411 (−1.2%)	2.265 (−0.1%)	2.136 (−0.4%)	1.877 (−1.3%)
K=2	MuLoCo	3.130 (+0.2%)	2.642 (+0.0%)	2.402 (+0.0%)	2.245 (+0.0%)	2.127 (−0.1%)	–
	DiLoCo	3.159 (+0.0%)	2.668 (−0.5%)	2.428 (−0.5%)	2.275 (+0.4%)	2.155 (+0.5%)	–
K=4	MuLoCo	3.148 (+0.8%)	2.651 (+0.4%)	2.410 (+0.4%)	2.254 (+0.4%)	2.139 (+0.5%)	–
	DiLoCo	3.192 (+1.1%)	2.697 (+0.6%)	2.447 (+0.3%)	2.283 (+0.7%)	2.166 (+1.0%)	–
K=8	MuLoCo	3.177 (+1.7%)	2.659 (+0.7%)	2.426 (+1.0%)	2.273 (+1.2%)	2.155 (+1.3%)	–
	DiLoCo	3.248 (+2.8%)	2.734 (+1.9%)	2.478 (+1.6%)	2.300 (+1.5%)	2.181 (+1.7%)	–
K=16	MuLoCo	3.222 (+3.1%)	2.713 (+2.8%)	2.448 (+1.9%)	2.291 (+2.0%)	2.165 (+1.7%)	1.902 (+1.4%)
	DiLoCo	3.326 (+5.3%)	2.808 (+4.7%)	2.522 (+3.4%)	2.348 (+3.6%)	2.215 (+3.3%)	1.910 (+0.4%)

Table 8 Zero-shot LM Evaluation Results all optimizers at 3.1B scale and DP, K=1, and K=16 at 15B scale. We report accuracy for MMLU, WinoGrande and normalized accuracy for HellaSwag, PIQA, ARC-E, ARC-C, OBQA. We additionally report mean evaluation accuracy across all available tasks.

Model Size	Configuration	MMLU	HellaSwag	WinoGrande	PIQA	ARC-E	ARC-C	OBQA	Mean Acc.
3.1B	DP AdamW	29.76%	66.98%	60.62%	76.33%	71.17%	42.66%	41.40%	55.56%
3.1B	DP Muon	37.24%	68.18%	62.95%	77.49%	71.19%	42.03%	42.47%	57.36%
3.1B	DiLoCo K=1	35.34%	67.58%	64.09%	77.58%	69.36%	41.81%	41.80%	56.79%
3.1B	DiLoCo K=2	27.46%	66.48%	61.40%	76.66%	69.23%	41.30%	40.80%	54.76%
3.1B	DiLoCo K=4	30.01%	65.84%	61.64%	76.82%	70.66%	41.30%	41.80%	55.44%
3.1B	DiLoCo K=8	28.18%	65.45%	60.38%	76.33%	70.08%	42.92%	39.40%	54.68%
3.1B	DiLoCo K=16	27.38%	63.03%	59.27%	75.95%	66.41%	41.72%	39.60%	53.34%
3.1B	MuLoCo K=1	37.10%	68.60%	63.54%	77.69%	71.42%	42.58%	42.60%	57.65%
3.1B	MuLoCo K=2	37.52%	68.35%	64.17%	77.69%	72.18%	42.24%	41.80%	57.71%
3.1B	MuLoCo K=4	35.43%	67.49%	61.01%	77.75%	70.20%	41.72%	43.40%	56.71%
3.1B	MuLoCo K=8	35.29%	66.82%	60.85%	77.31%	70.45%	43.00%	41.40%	56.45%
3.1B	MuLoCo K=16	29.50%	65.46%	60.38%	75.90%	68.01%	39.68%	42.40%	54.48%
15B	DP AdamW	57.20%	77.69%	71.19%	81.39%	78.11%	52.30%	45.40%	66.18%
15B	DP Muon	59.14%	79.05%	71.19%	81.12%	77.23%	53.07%	46.80%	66.80%
15B	DiLoCo K=1	59.80%	79.06%	71.43%	81.34%	77.99%	52.56%	46.40%	66.94%
15B	DiLoCo K=16	54.15%	77.59%	69.30%	81.72%	77.65%	52.56%	46.80%	65.68%
15B	MuLoCo K=1	59.49%	79.06%	71.11%	81.61%	78.11%	52.73%	46.60%	66.96%
15B	MuLoCo K=16	56.96%	77.18%	70.72%	80.74%	76.60%	53.16%	46.20%	65.94%

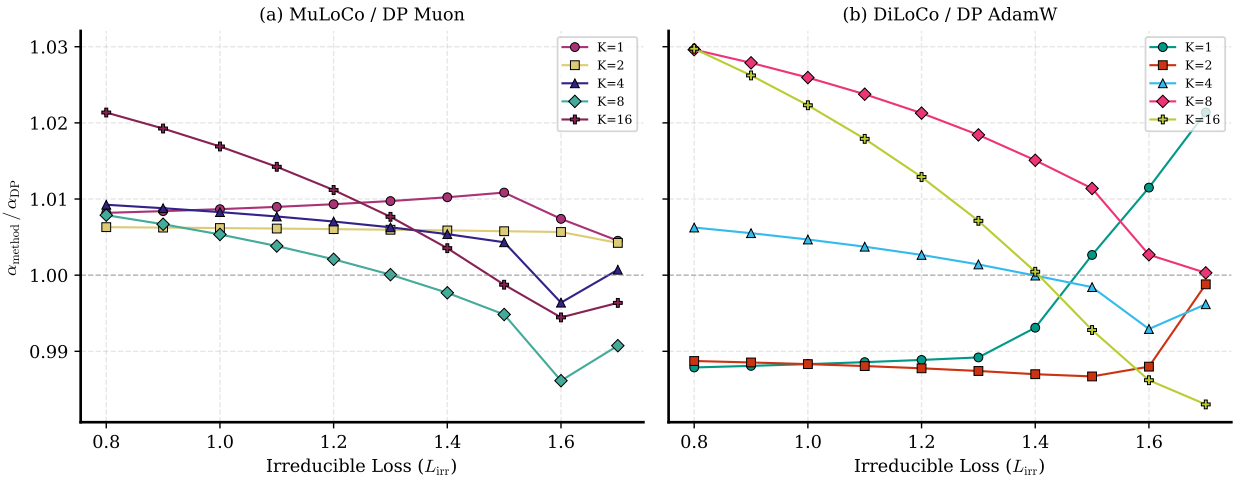
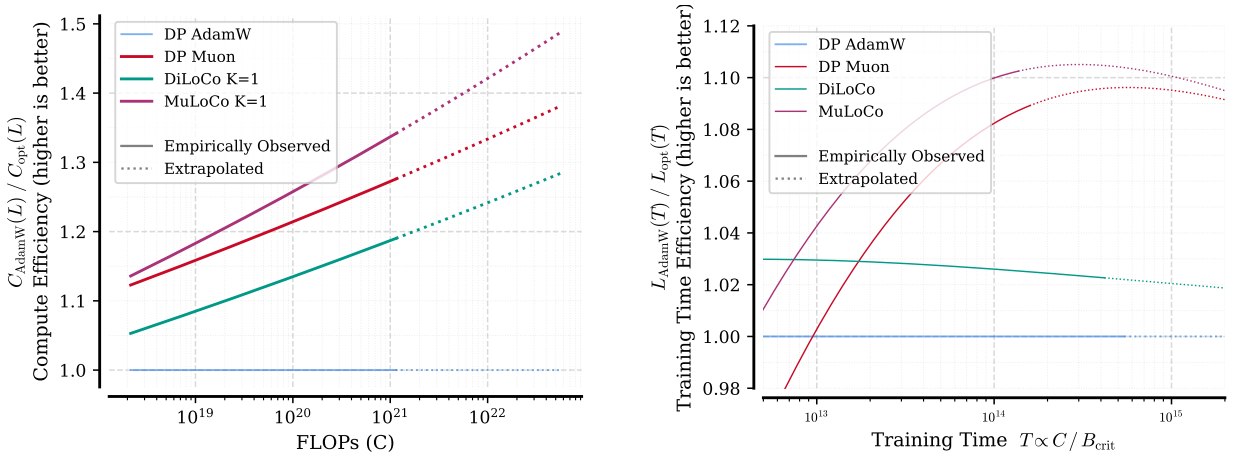


Figure 17 Scaling exponents as a function of the assumed irreducible loss. We fit compute scaling laws of the form $L(C) = aC^{\alpha} + L_{\text{irr}}$ to our 150M-3.1B datapoints while holding L_{irr} fixed to a predetermined value that is shared across all methods. For each L_{irr}, we report the resulting scaling exponent α *normalized by the corresponding data-parallel (DP) baseline* (y-axis: $\alpha_{\text{method}}/\alpha_{\text{DP}}$), so values above 1 indicate better than DP scaling while values below 1 indicate worse scaling than DP. The left panel shows MuLoCo normalized by DP Muon, while the right panel shows DiLoCo normalized by DP AdamW. At lower irreducible losses, the fits suggest that higher-worker MuLoCo configurations (larger K) become increasingly competitive and will catch up to and surpass DP performance at scale.



(a) Compute efficiency at optimal batch size

(b) Normalized performance at iso training time

Figure 18 MuLoCo improves both compute efficiency and training-time-normalized performance. (a) We report iso-loss compute efficiency, $C_{\text{AdamW}}(L)/C_{\text{opt}}(L)$, using scaling laws fit at each method's optimal batch size. Values above one indicate that optimizer m reaches the same loss as AdamW using fewer FLOPs. (b) We report normalized performance at fixed training time, where training time is measured as FLOPs divided by the critical batch size. This metric is evaluated at B_{crit} and assumes enough accelerators are scaled in proportion to the critical batch size to realize the corresponding reduction in sequential computation. Under this assumption, MuLoCo achieves an ∼8–10% loss improvement over AdamW for the same training-time budget. Solid curves indicate the empirically observed regime, while dotted curves indicate extrapolation beyond the range of available data.

Compute efficiency and training-time-normalized performance. Figure 18 provides two complementary views of the efficiency gains implied by our scaling laws. The compute-efficiency plot uses the same compute–loss scaling-law methodology as Sec. 7.1, fit across the extensively swept 150M–3.1B models, but evaluates each optimizer at its optimal batch size. For a target loss L, we invert the fitted loss curve for each optimizer to obtain the compute required to reach that loss, $C_m(L)$, and report $C_{\text{AdamW}}(L)/C_{\text{opt}}(L)$. Values above one therefore, indicate that optimizer m reaches the same loss using fewer FLOPs than AdamW. In contrast, the training-time-normalized performance plot evaluates each method at its critical batch size, using $T \propto C/B_{\text{crit}}(C)$ as a proxy for the amount of sequential computation required. This asks a different question: given the same training-time budget, how much better is the final loss if enough accelerators are used in parallel to scale with the method's critical batch size? We find that MuLoCo improves over AdamW under both views: it is more compute-efficient at the optimal batch size ($\sim 1.35\times$), and when training at B_{crit}, it can achieve an ∼8–10% loss improvement over AdamW for the same training-time budget.

C.3 Extended 15B results

The following section reports on a number of additional results from our 15B experiments. First we present timings for the 15B training runs with Muon and AdamW (Table 9). We find that the additional overhead of Muon's optimizer step is negligible at 15B scale. Using these metrics, we provide idealized wall-clock training estimates for different bandwidth networks, assuming the GPUs are scaled proportionally to batch size (Table 10 and Figures 20). Finally, we also present validation loss curves for these training runs as a function of training tokens and training steps (Figure 19).

Metric	AdamW	Muon	Relative Δ (%)
End-to-End (s)	0.9753	0.9832	+0.81
MFU (%)	39.89	39.57	−0.80
TFLOPS/GPU	394.54	391.37	−0.80
Throughput (tokens/s/GPU)	4,200	4,166	−0.81
Memory Complexity (Parameter copies)	4	3	−25

Table 9 System-level performance comparison of training with AdamW and Muon at 15B scale. End-to-end step time when training at local batch size 2 and global batch size 1024 with 16 FSDP ranks and 32 data parallel ranks. Metrics correspond to the median over the first 500 steps. Relative Δ computed as $\frac{\text{Muon}-\text{AdamW}}{\text{AdamW}} \times 100$. We observe that the additional overhead of Muon's optimizer step is negligible at 15B scale.

Table 10 Idealized wall-clock training time under bandwidth constraints. We report idealized wall-clock times (in hours) estimated using the 15B system measurements in 9. The estimates factor in global batch size, communication overhead, optimizer step time, and forward/backward (FW/BW) compute time. We observe that communication-efficient optimizers dominate at low bandwidth, while larger batch sizes dominate under high bandwidth.

Method	Training Time (hours)					
	10 Gbit/s	100 Gbit/s	400 Gbit/s	1,600 Gbit/s	3,200 Gbit/s	6,400 Gbit/s
Data-parallel AdamW (BS=2.1M)	1008.2	136.3	63.6	45.4	42.4	40.9
Data-parallel Muon (BS=4.2M)	504.3	68.3	32.0	22.9	21.4	20.6
K=1 DiLoCo (BS=1M)	2016.3	272.5	127.2	90.8	84.8	81.8
K=1 MuLoCo (BS=16.8M)	126.1	17.1	8.0	5.7	5.3	5.2
K=16 DiLoCo (BS=4.2M)	35.8	21.3	20.1	19.8	19.7	19.7
K=16 MuLoCo (BS=8.4M)	18.0	10.7	10.1	10.0	9.9	9.9

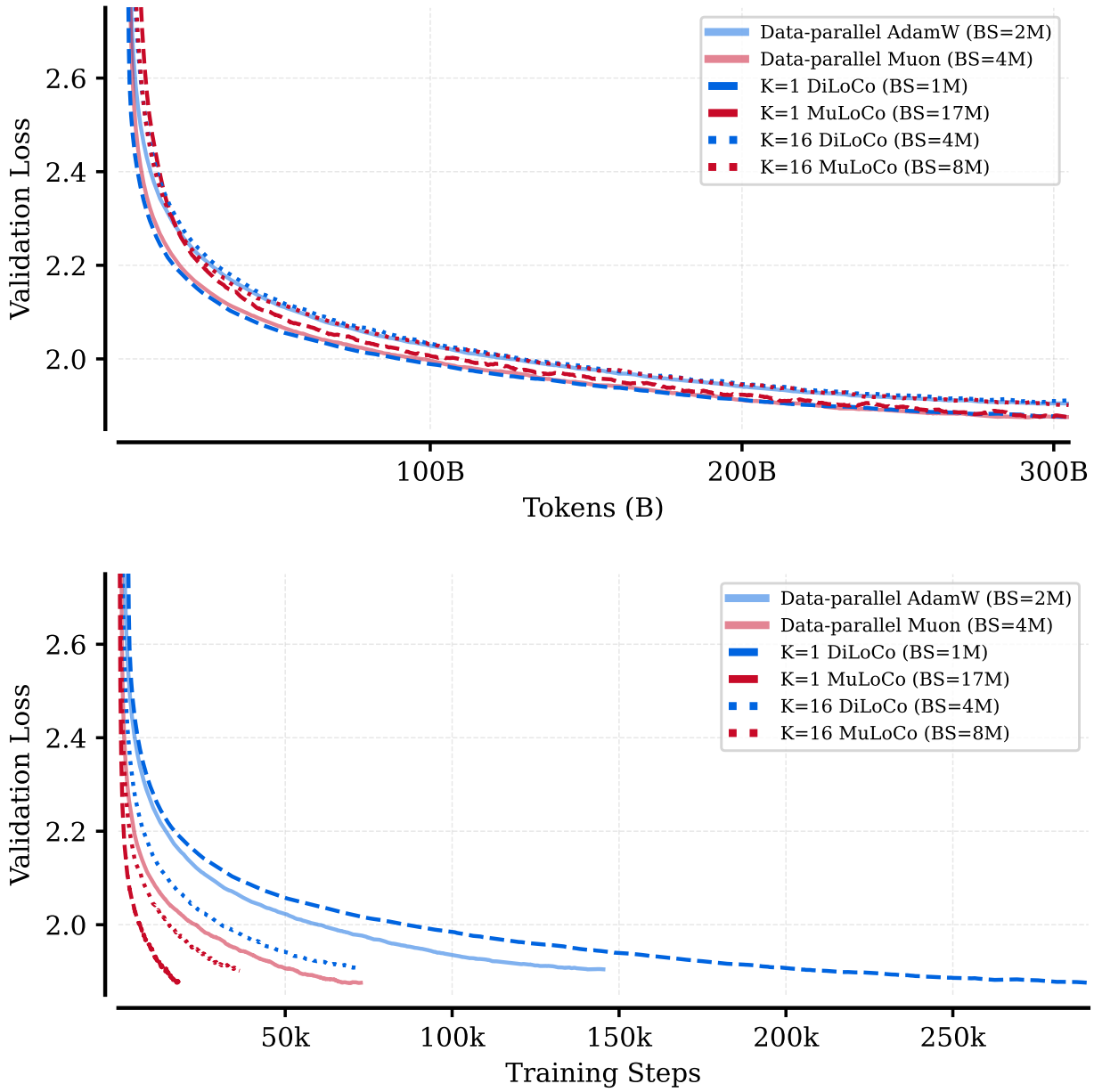


Figure 19 Validation loss for 15B pre-training runs. We extrapolate optimal hyperparameters to 15B scale and train models with different optimizers. *Top:* validation loss as a function of training tokens. *Bottom:* validation loss as a function of optimization steps. Despite using a **16× larger batch size**, MuLoCo ($K = 1$, BS=17M) reaches nearly the same final loss as DiLoCo ($K = 1$, BS=1M), demonstrating its strong capability for large-batch training. For $K = 16$, DiLoCo narrows the gap to data-parallel AdamW, but both remain notably worse than data-parallel Muon, MuLoCo K=1, and DiLoCo K=1. Despite reaching nearly the same loss as DP AdamW and DiLoCo K=16, MuLoCo $K = 16$ can leverage larger batch sizes.

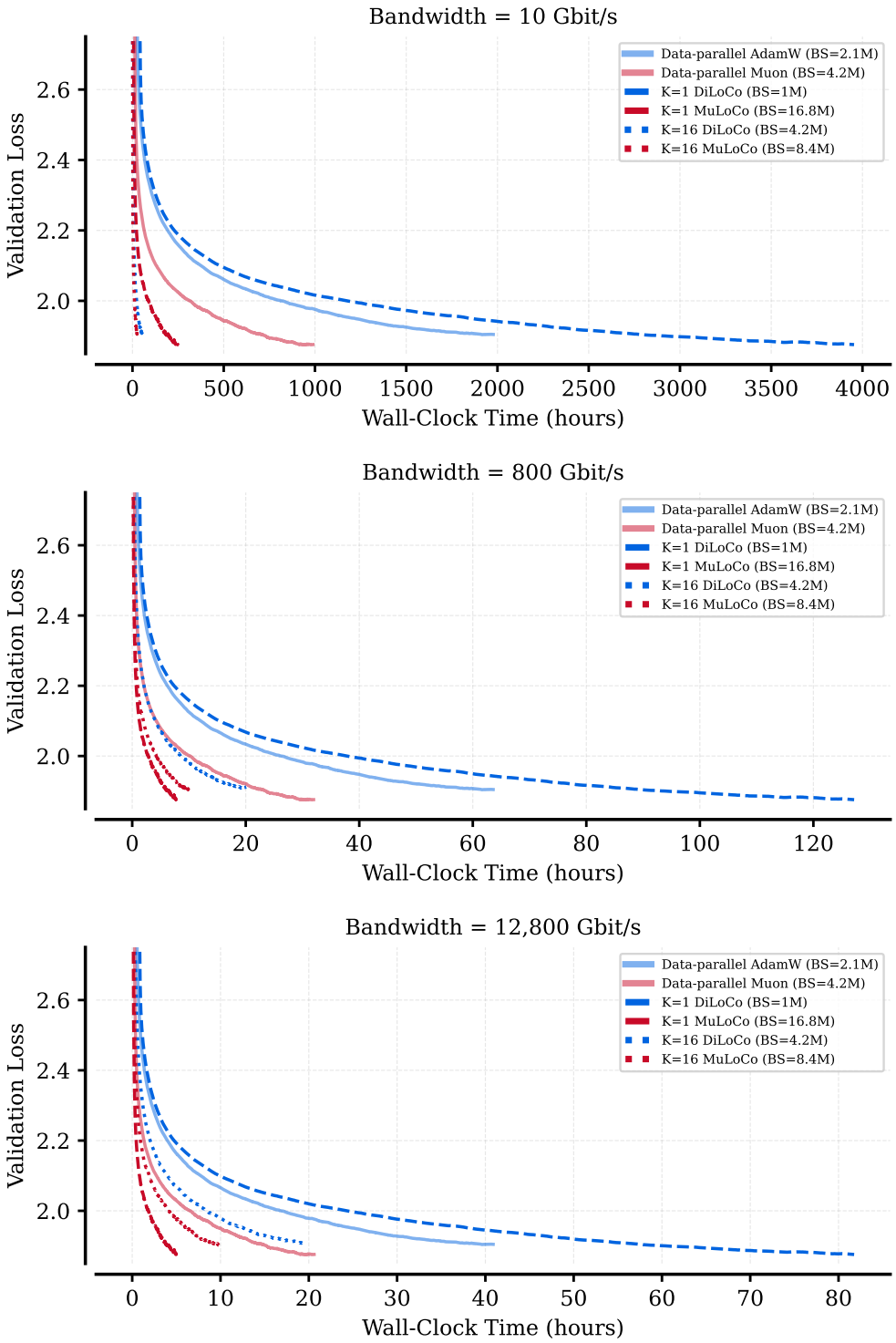
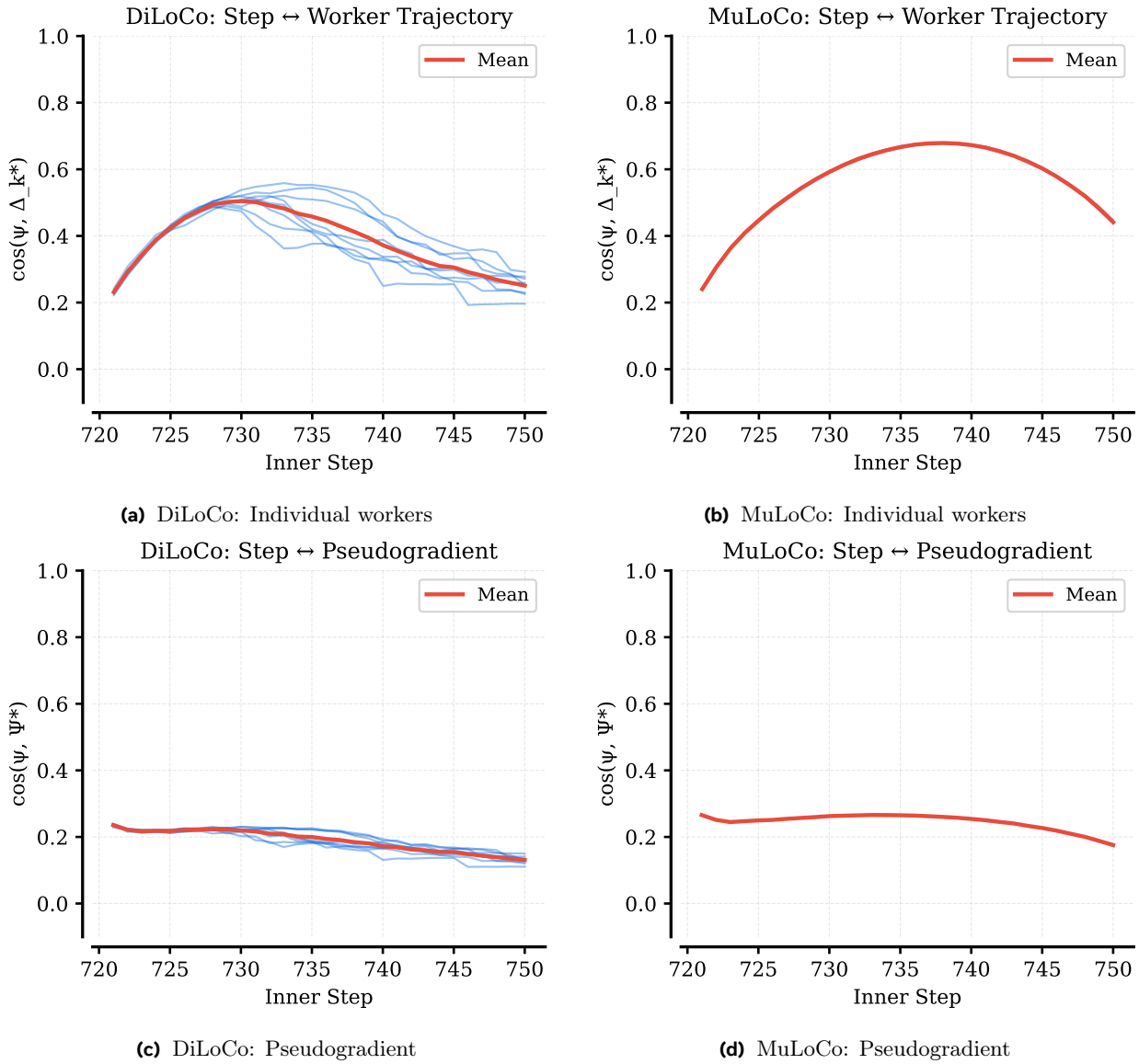


Figure 20 Training time under bandwidth constraints. We compare validation loss versus wall-clock time across network bandwidths of 10, 800, and 12,800 Gbit/s under the assumption that the cluster size is scaled proportional to the batch size. Both batch size and communication time critically influence overall wall-clock training time. In the bandwidth-constrained regime (10 Gbit/s), batch size still matters, but reducing communication frequency dominates: the K=16 optimizers train substantially faster than K=1 MuLoCo despite using smaller batch sizes. As network speed increases, communication becomes less of a bottleneck and larger batch sizes become more advantageous. At 12,800 Gbit/s, we observe that K=1 MuLoCo (with its larger batch size) trains faster, illustrating the shifting trade-off between communication cost and statistical efficiency.

D Extended pseudogradient analysis



(a) DiLoCo: Individual workers

(b) MuLoCo: Individual workers

(c) DiLoCo: Pseudogradient

(d) MuLoCo: Pseudogradient

Figure 21 Worker-level variability in step alignment for DiLoCo vs. MuLoCo. Each plot overlays the trajectories of *individual workers* together with their *mean* (labeled "Mean"). For MuLoCo (right column), the mean curve lies almost exactly on top of the individual-worker curves, making them largely indistinguishable. In contrast, DiLoCo (left column) exhibits visibly separated worker trajectories, indicating substantially higher variance across workers. Top row shows individual-worker trajectories; bottom row shows the corresponding pseudogradient views.

E Optimal hyperparameters

E.1 Hyperparameter Sweep Details

As described in Section 5, we conduct extensive hyperparameter sweeps at each model scale for all optimizers and worker counts. In total, our scaling study comprises over 2,200 individual pre-training runs across six model scales (150M–15B), four optimizers (DP AdamW, DP Muon, DiLoCo, MuLoCo), and five worker counts ($K \in \{1, 2, 4, 8, 16\}$). We summarize the sweep ranges, tuning strategy, and optimal hyperparameters found below.

Table 11 Hyperparameter search spaces. Ranges are aggregated across all model scales and worker counts. They correspond to the ranges that we needed to sweep in order to ensure the optimal HP value was an interior point of the values tried. Learning rates follow a $\sqrt{2}$ grid; batch sizes follow a powers-of-2 grid. Weight decay is rescaled with batch size per Wang and Aitchison (2024). Outer optimizer HPs are swept at 416M and reused at larger scales.

Hyperparameter	DP AdamW	DP Muon	DiLoCo	MuLoCo
Inner LR (η_{in})	[1.7e-4, 6.3e-2]	[3.9e-3, 3.5e-1]	[3.5e-4, 8.8e-2]	[4.9e-4, 1.8e-1]
Weight decay (λ)	[6.3e-5, 1e-1]	[1e-4, 1e-1]	[1.6e-5, 1.7e-1]	[5e-5, 8e-2]
Global batch size (B)	{32–4096}	{32–8192}	{8–8192}	{8–8192}
Outer LR (η_{out})	—	—	{0.4–1.1}	{0.4–1.1}
Outer momentum (μ)	—	—	{0.5–0.9}	{0.5–0.9}
Total runs (all scales)	339	260	822	808

Sweep ranges. Table 11 summarizes the hyperparameter search spaces used across our scaling study. All learning rates are swept on a multiplicative grid of powers of $\sqrt{2}$ (i.e., each value is $\sqrt{2}\times$ the previous), while batch sizes are swept on powers of 2. Weight decay values are rescaled according to Wang and Aitchison (2024) as the batch size changes: for DP methods we rescale by the global batch size, and for DiLoCo/MuLoCo we rescale by the per-worker batch size (B/K). The outer optimizer hyperparameters (η_{out}, μ) are extensively swept at the 416M scale and reused at other scales, following Charles et al. (2025)'s finding that they remain roughly constant across model sizes.

Optimal hyperparameters at each scale. Tables 12–14 report the optimal hyperparameters found at each model scale. Several trends emerge:

- **Learning rate decreases with scale** for AdamW-based methods (DP AdamW, DiLoCo), dropping roughly 30× from 150M to 3.1B. In contrast, Muon-based methods (DP Muon, MuLoCo) maintain relatively stable learning rates across scales.
- **Weight decay is tied to batch size** via the rescaling rule of Wang and Aitchison (2024). Within a fixed scale, λ decreases as K increases because the per-worker batch size (B/K) decreases.
- **Batch size grows with scale** for all methods. DP methods and K=1 workers generally use $B = 32$–256 at 150M–3.1B. At higher K, the per-worker batch size decreases while the total token batch size ($B \times K \times 2048$) remains roughly constant within a scale.
- **MuLoCo favors lower outer momentum at small K.** As shown in Figure 22, MuLoCo uses $\mu = 0.6$ at $K = 1$ versus $\mu = 0.8$ for DiLoCo. At $K \geq 4$, both converge to $\mu = 0.8$–0.9.
- **Outer learning rate increases with K** for both DiLoCo and MuLoCo, rising from $\eta_{\text{out}} = 0.6$–0.7 at $K = 1$ to $\eta_{\text{out}} = 1.0$ at $K = 16$.

Outer optimizer hyperparameters. For both DiLoCo and MuLoCo, we observe a consistent pattern across all model sizes: (1) the outer learning rate increases from 0.6–0.7 at $K = 1$ to 1.0 at $K = 16$, and (2) the outer momentum increases from 0.6–0.8 at $K = 1$ to 0.9 at $K \geq 8$. Notably, MuLoCo uses a lower outer momentum ($\mu = 0.6$) than DiLoCo ($\mu = 0.8$) at $K = 1$, which is consistent with Muon's optimizer steps already being well-aligned (see Section 6.1), reducing the benefit of additional momentum. These values are extensively tuned at the 416M scale (Table 11) and reused at all other scales. For a visualization of optimal values, see Figure 22

Weight decay rescaling. At each model scale, we first determine the optimal weight decay λ^* for DP AdamW and DP Muon at a fixed batch size of $B = 1$M tokens (approximately $B = 512$ sequences). When sweeping other batch sizes, we rescale λ^* following Wang and Aitchison (2024). For DiLoCo and MuLoCo, the weight decay is rescaled according to the per-worker batch size (B/K), which explains the systematic decrease of λ with increasing K observed in Tables 13 and 14.

Table 12 Optimal hyperparameters for data-parallel baselines at each model scale. η_{in}: inner learning rate; λ: weight decay; B: global batch size (sequences); B_{tok}: global token batch size ($= B \times 2048$). Since $K = 1$ for DP methods, $B/K = B$.

Scale	Method	η_{in}	λ	B	B_{tok}
150M	DP AdamW	4.42e-2	1.25e-4	64	128K
	DP Muon	1.25e-1	1.25e-3	64	128K
416M	DP AdamW	1.10e-2	3.00e-2	512	1M
	DP Muon	8.84e-2	2.50e-3	512	1M
914M	DP AdamW	5.52e-3	5.00e-3	256	512K
	DP Muon	3.13e-2	5.00e-3	256	512K
1.76B	DP AdamW	1.38e-3	1.00e-1	512	1M
	DP Muon	3.13e-2	1.00e-3	512	1M
3.07B	DP AdamW	1.38e-3	5.00e-4	256	512K
	DP Muon	2.21e-2	1.00e-3	512	1M
15.2B	DP AdamW	1.73e-4	2.00e-4	1024	2M
	DP Muon	1.10e-2	4.00e-4	2048	4M

Table 13 Optimal hyperparameters for DiLoCo. At each model scale and worker count K. B/K: per-worker batch size (sequences); B: global batch size; B_{tok}: global token batch size ($= B \times 2048$). All runs use $H = 30$ synchronization steps and SGD with Nesterov momentum as the outer optimizer.

Scale	K	η_{in}	λ	B/K	B	B_{tok}	η_{out}	μ
150M	1	2.21e-2	2.50e-4	128	128	256K	0.6	0.8
	2	4.42e-2	1.25e-4	64	128	256K	0.9	0.8
	4	2.21e-2	6.25e-5	32	128	256K	0.9	0.8
	8	4.42e-2	3.13e-5	16	128	256K	0.9	0.9
	16	2.21e-2	3.13e-5	16	256	512K	1.0	0.9
416M	1	1.10e-2	5.00e-3	256	256	512K	0.6	0.8
	2	1.10e-2	2.50e-3	128	256	512K	0.9	0.8
	4	1.10e-2	1.25e-3	64	256	512K	0.9	0.8
	8	7.81e-3	1.25e-3	64	512	1M	0.7	0.9
	16	2.76e-3	6.25e-4	32	512	1M	1.0	0.9
914M	1	5.52e-3	5.00e-3	256	256	512K	0.6	0.8
	2	3.91e-3	2.50e-3	128	256	512K	0.9	0.8
	4	3.91e-3	1.25e-3	64	256	512K	0.9	0.8
	8	2.76e-3	1.25e-3	64	512	1M	0.9	0.9
	16	1.95e-3	6.25e-4	32	512	1M	1.0	0.9
1.76B	1	1.95e-3	1.00e-3	512	512	1M	0.6	0.8
	2	1.38e-3	5.00e-4	256	512	1M	0.9	0.8
	4	1.95e-3	2.50e-4	128	512	1M	0.9	0.8
	8	1.38e-3	1.25e-4	64	512	1M	0.9	0.9
	16	1.38e-3	6.25e-5	32	512	1M	1.0	0.9
3.07B	1	1.38e-3	5.00e-4	256	256	512K	0.6	0.8
	2	6.91e-4	5.00e-4	256	512	1M	0.9	0.8
	4	1.38e-3	2.50e-4	128	512	1M	0.9	0.8
	8	1.38e-3	1.25e-4	64	512	1M	0.9	0.9
	16	6.91e-4	1.25e-4	64	1024	2M	1.0	0.9

Table 14 Optimal hyperparameters for MuLoCo. At each model scale and worker count K. B/K: per-worker batch size (sequences); B: global batch size; B_{tok}: global token batch size ($= B \times 2048$). All runs use $H = 30$ synchronization steps and SGD with Nesterov momentum as the outer optimizer.

Scale	K	η_{in}	λ	B/K	B	B_{tok}	η_{out}	μ
150M	1	6.25e-2	1.25e-3	64	64	128K	0.7	0.6
	2	8.84e-2	1.25e-3	64	128	256K	0.9	0.7
	4	8.84e-2	6.25e-4	32	128	256K	0.9	0.8
	8	1.25e-1	6.25e-4	32	256	512K	0.9	0.8
	16	6.25e-2	3.13e-4	16	256	512K	1.0	0.9
416M	1	4.42e-2	5.00e-3	256	256	512K	0.7	0.6
	2	4.42e-2	5.00e-3	256	512	1M	0.9	0.7
	4	4.42e-2	2.50e-3	128	512	1M	0.9	0.8
	8	6.25e-2	1.25e-3	64	512	1M	0.9	0.8
	16	4.42e-2	6.25e-4	32	512	1M	1.0	0.9
914M	1	2.21e-2	5.00e-3	256	256	512K	0.7	0.6
	2	2.21e-2	2.50e-3	128	256	512K	0.9	0.7
	4	2.21e-2	1.25e-3	64	256	512K	0.9	0.8
	8	3.13e-2	1.25e-3	64	512	1M	0.9	0.8
	16	3.13e-2	1.25e-3	64	1024	2M	1.0	0.9
1.76B	1	3.13e-2	2.00e-3	1024	1024	2M	0.7	0.6
	2	2.21e-2	5.00e-4	256	512	1M	0.9	0.7
	4	2.21e-2	2.50e-4	128	512	1M	0.9	0.8
	8	1.56e-2	1.25e-4	64	512	1M	0.9	0.8
	16	1.56e-2	1.25e-4	64	1024	2M	1.0	0.9
3.07B	1	1.56e-2	5.00e-4	256	256	512K	0.7	0.6
	2	1.56e-2	5.00e-4	256	512	1M	0.9	0.8
	4	1.56e-2	2.50e-4	128	512	1M	0.9	0.8
	8	1.56e-2	2.50e-4	128	1024	2M	0.9	0.8
	16	1.56e-2	2.50e-4	128	2048	4M	1.0	0.9

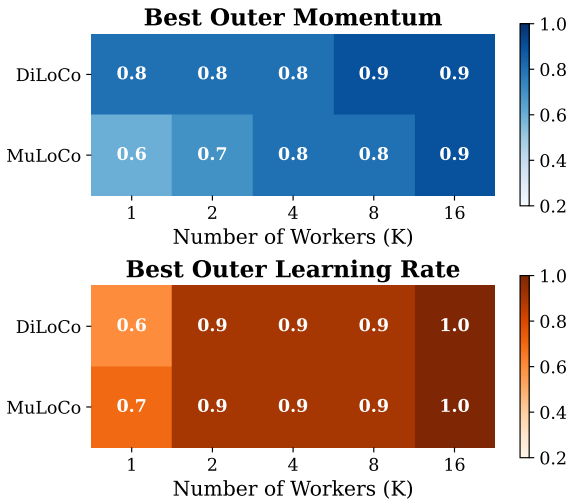


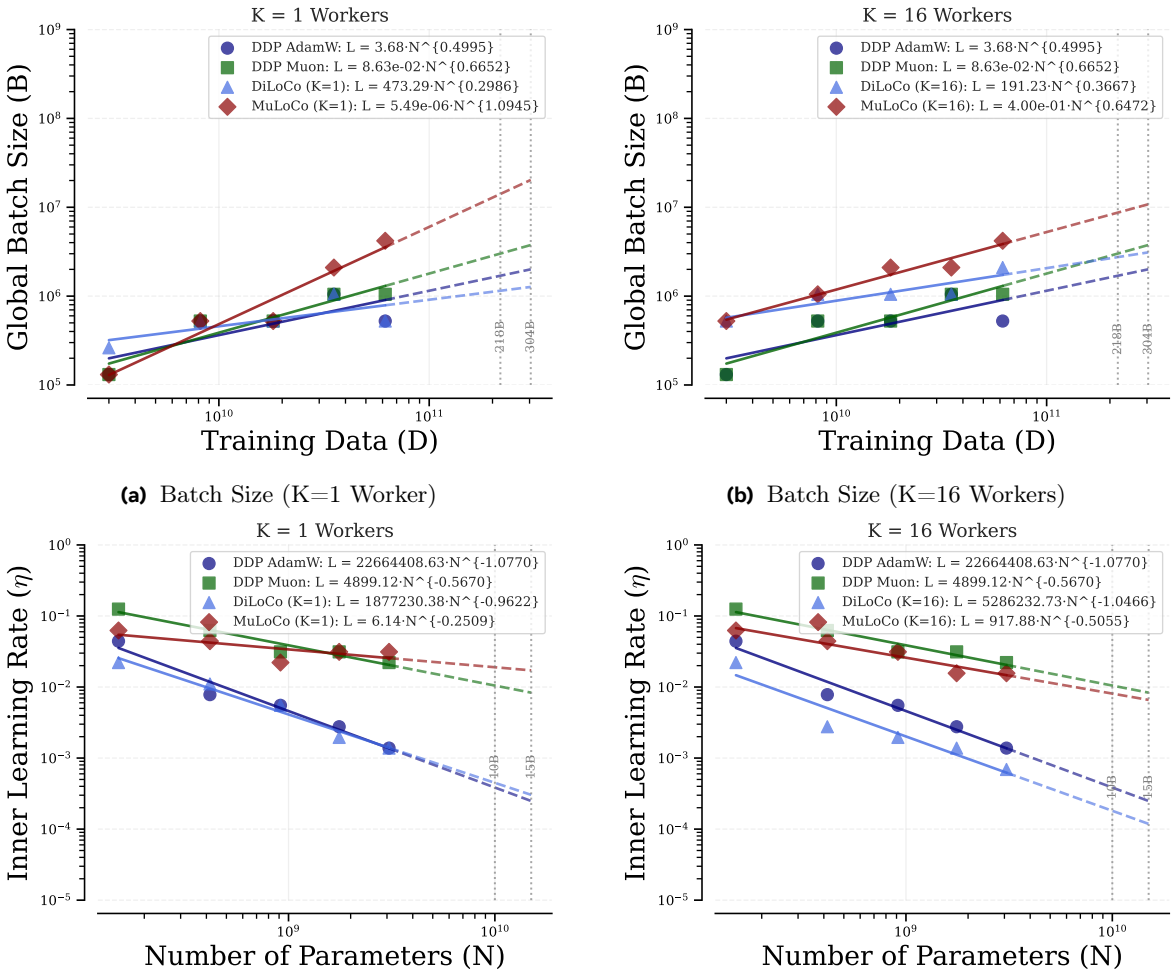
Figure 22 At low worker counts MuLoCo's outer momentum should be decreased. When tuning the outer optimizer hyperparameters for MuLoCo and DiLoCo, we found that MuLoCo favors lower momentum coefficients at lower worker counts relative to DiLoCo.

E.2 Extropolating HPs to 15B scale

At the 15B scale, running full hyperparameter sweeps is computationally prohibitive. Instead, we extrapolate optimal weight decay, inner learning rate, and batch size from power law fits to smaller scales. To promote better accelerator utilization, we use the nearest power-of-2 batch size to the optimal extrapolated value. Depending on whether we rounded the batch size up or down, we adjust the learning rate up or down, respectively to its nearest $\sqrt{2}$ power. For weight decay, we used $1e-4$ for both AdamW and Muon at $B = 1$M since the extrapolated values were close enough, and this matches our other experiments. The hyperparameters used are reported in table 15. It should be noted that our power laws with B^* were fit with K=1 MuLoCo, having $B^* = 4$M at 3.1B scale. However, further sweeping later revealed that this was a local minimum (see Fig. 12). This means that K=1 MuLoCo's batch size is likely closer to the CBS than B^* at 15B scale and that its optimal final performance is likely slightly under-reported. The power law fits used for B^* and η are reported in Figure. 23.

Table 15 Hyperparameters for 15B runs. These are extrapolated from power law fits to smaller scales (150M–3.07B) with minor manual adjustments. B/K: per-worker batch size (sequences); B: global batch size (sequences); B_{tok}: global token batch size ($= B \times 2048$).

Method	K	η_{in}	λ	B/K	B	B_{tok}	η_{out}	μ
DP AdamW	1	1.73e-4	2.00e-4	1024	1024	2M	—	—
DP Muon	1	1.10e-2	4.00e-4	2048	2048	4M	—	—
DiLoCo	1	3.45e-4	1.00e-4	512	512	1M	0.6	0.8
DiLoCo	16	3.45e-4	2.50e-5	128	2048	4M	1.0	0.9
MuLoCo	1	1.56e-2	1.60e-3	8192	8192	16M	0.7	0.6
MuLoCo	16	1.10e-2	5.00e-5	256	4096	8M	1.0	0.9



(a) Batch Size (K=1 Worker)

(b) Batch Size (K=16 Workers)

(c) Learning Rate (K=1 Worker)

(d) Learning Rate (K=16 Workers)

Figure 23 Extrapolating optimal batch size (B^*) and inner learning rate (η_{in}) for different optimization algorithms. We fit a power law in tokens (D) to the optimal batch size and in parameters (N) for the optimal learning rates at each scale. ****NOTE 1:** These were fit before realizing that B=4M was a local minimum for K=1 MuLoCo at 3.1B scale, so the optimal batch size is slightly larger. We still chose to report these fits since they were used in our 15B experiments. ****NOTE 2:** The legend in these plots shows an powerlaw with misleading variables. It should be $B = AD^{\alpha}$ for (a,b) $\eta_{\text{in}} = AN^{\alpha}$ for (c,d). The y and x axes are correct.

F A Robuts Evaluation Loss Estimate

Throughout this work, we evaluate each training run by computing a smoothed evaluation loss $\hat{L}$ from its validation loss trajectory. Directly using the final recorded validation loss is unreliable for comparing runs and fitting scaling laws, as validation losses are logged at discrete intervals and exhibit non-negligible noise, overestimating or underestimating final performance (see Figure 24). This is likely caused by sampling different final evaluations depending on the number of training iterations for the run in question. We therefore adopt a time-weighted exponential moving average (EMA) that accounts for the temporal spacing between measurements and provides a smooth, robust estimate of the final performance.

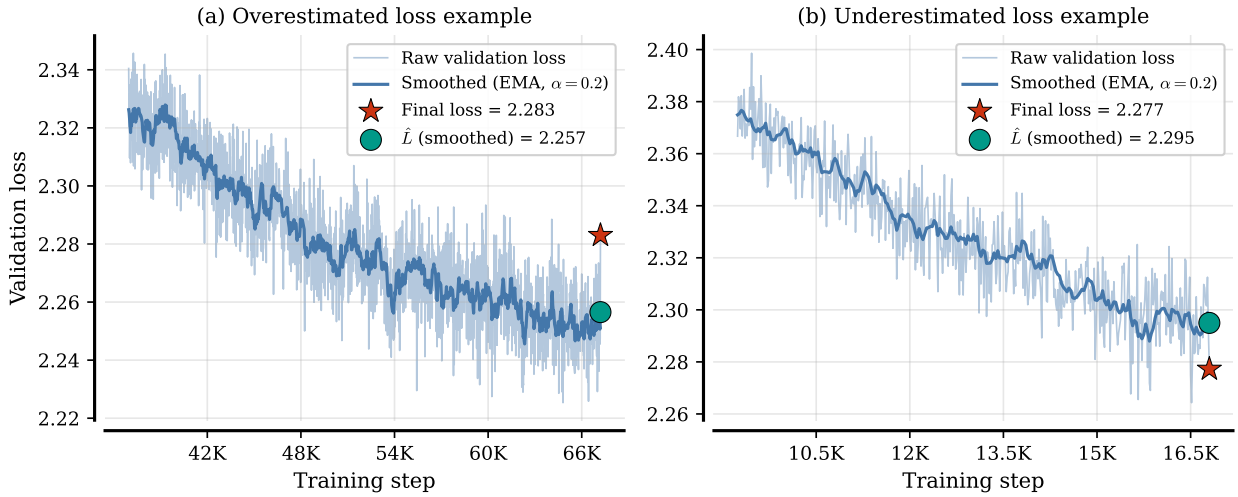


Figure 24 Motivation for reporting a smoothed evaluation loss $\hat{L}$ rather than the final logged validation loss. Each panel shows a real validation-loss trajectory near the end of training (raw points) together with the time-weighted EMA used in Sec. F (with $\alpha = 0.2$ and filtering to synchronization boundaries at $H = 30$). **Left:** an *overestimated* final loss, where the last evaluated batch happens to be unusually difficult (the final logged loss exceeds $\hat{L}$). **Right:** an *underestimated* final loss, where the last evaluated batch is unusually easy (the final logged loss is below $\hat{L}$). These examples illustrate that directly using the final recorded validation loss can introduce noise-driven bias in run comparisons and scaling-law fits, whereas $\hat{L}$ provides a more robust final performance estimate.

Concretely, each run records a validation loss trajectory $\{(\ell_i, t_i)\}_{i=1}^{N}$, where ℓ_i is the validation loss (cross-entropy on 2M token batches from the NemotronCC held-out split) measured at training step t_i. Since DiLoCo and MuLoCo synchronize worker parameters every H inner steps, we first filter the trajectory to retain only measurements taken at synchronization boundaries, i.e., steps t_i satisfying $t_i \bmod H = 0$ (with $H = 30$ throughout this work). This filtering ensures that all methods—including data-parallel baselines—are evaluated at steps where the model parameters correspond to a post-synchronization state rather than an intermediate local state. For data-parallel methods, which effectively synchronize every step, this subsampling still produces a dense set of measurements.

After filtering, we obtain a subsequence $\{(\ell_j, t_j)\}_{j=1}^{M}$ and apply a time-weighted EMA. Unlike a standard EMA that assigns a fixed weight α to each new observation, we adjust the effective smoothing factor based on the elapsed time $\Delta t_j = t_j - t_{j-1}$ between consecutive measurements. The smoothed sequence $\{s_j\}_{j=1}^{M}$ is computed as

$$s_1 = \ell_1, \qquad s_j = \tilde{\alpha}_j \, \ell_j + (1 - \tilde{\alpha}_j) \, s_{j-1}, \quad j = 2, \ldots, M, \tag{10}$$

where the adaptive smoothing coefficient is

$$\tilde{\alpha}_j = 1 - \exp\Big(-\frac{\alpha \, \Delta t_j}{H}\Big). \tag{11}$$

Here $\alpha > 0$ is a base smoothing parameter. When measurements are evenly spaced at every H steps ($\Delta t_j = H$), the adaptive coefficient reduces to the constant $\tilde{\alpha} = 1 - e^{-\alpha}$. The evaluation loss for the run is then $\hat{L} = s_M$, the final value of the smoothed sequence.

We use $\alpha = 0.2$ throughout, which yields $\tilde{\alpha} \approx 0.181$ at the nominal spacing of $H = 30$ steps. This provides substantial smoothing (an effective window spanning roughly the last $1/\tilde{\alpha} \approx 5$–6 synchronization intervals) while remaining responsive enough to track the loss trajectory near convergence. All scaling law fits, hyperparameter comparisons, and optimal run selections reported in this paper use this smoothed evaluation loss $\hat{L}$.