MuLoCo: Muon is a practical inner optimizer for DiLoCo

Benjamin Thérien [1 2] **Xiaolong Huang** [3 2] **Irina Rish** [1 2] **Eugene Belilovsky** [3 2]

Abstract

DiLoCo is a powerful framework for training large language models (LLMs) under networking constraints with advantages for increasing parallelism and accelerator utilization in data center settings. Despite significantly reducing communication frequency, however, DiLoCo's communication steps still involve all-reducing a complete copy of the model's parameters. While existing works have explored ways to reduce communication in DiLoCo, the role of error feedback accumulators and the effect of the inner-optimizer on compressibility remain under-explored. In this work, we investigate the effectiveness of standard compression methods—including Top-k sparsification and quantization for reducing the communication overhead of DiLoCo when paired with two local optimizers (AdamW and Muon). Our experiments pre-training decoder-only transformer language models (LMs) reveal that leveraging Muon as the inner optimizer for DiLoCo along with an error-feedback accumulator allows to aggressively compress the communicated delta to 2 bits with next to no performance degradation. Crucially, MuLoCo (Muon inner optimizer DiLoCo) significantly outperforms DiLoCo while communicating $8\times$ less and having identical memory complexity.

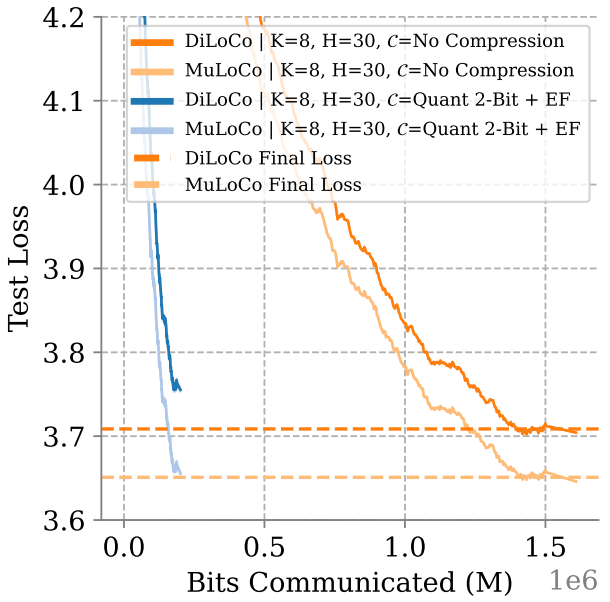


Figure 1: **MuLoCo (Muon inner optimizer) with 2-bit quantization and error feedback outperforms standard AdamW-DiLoCo with $8\times$ less communication while having identical memory complexity.** The figure reports the test loss (y-axis) measured at each communication step during pre-training. The x-axis reports the total number of bits communicated for a 220M parameter transformer LM. We use $K = 8$ workers and $H = 30$ local steps.

1. Introduction

It is now well established that increasing the model and dataset size improves performance of foundation models. Under this paradigm, training state-of-the-art models is infeasible on a single accelerator. Distributed computation across multiple accelerators is, therefore, required. However, standard data parallel training incurs communication costs proportional to the model size at each optimization step. For large models or slow networks, communication can become a significant bottleneck, leading to the development of many communication-efficient training algorithms.

Common approaches to reduce communication costs generally fall into three categories: those that reduce the frequency of communication, decrease the size of data communicated, or combine both approaches. The idea of reducing communication frequency in favor of more local computation was originally popularized in the context of federated learning, where it is known as FedAVG (McMahan et al., 2017). Local SGD (Stich, 2018) is an equivalent algorithm for non-federated settings. More recently, DiLoCo (Douillard et al., 2023) (a variant of Local SGD) has recently gained attention due to its competitive performance with data-parallel pre-training of LLMs and larger effective batch sizes (Charles et al., 2025). Another direction for reducing communication is to compress the com-

[1]DIRO, Université de Montréal, Montréal, Canada [2]Mila – Quebec AI Institute, Montréal, Canada [3]Concordia University, Montréal, Canada. Correspondence to: Benjamin Thérien <benjamin.therien@umontreal.ca>.

Efficient Systems for Foundation Models (ES-FoMo) Workshop, 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

municated updates (Wang et al., 2023; Ahn & Xu, 2025; Peng et al., 2024; Vogels et al., 2019). These approaches are typically combined with an error feedback accumulator (Karimireddy et al., 2019) to persist the information lost during compression and provide convergence guarantees. Finally, (Basu et al., 2019) and (Douillard et al., 2025) combine reduced communication frequency with smaller communication sizes, but neither studies compression with error feedback in the context of LLMs pre-training nor do they consider the influence of the local optimizers updates on compressibility.

Table 1: **Memory complexity of DiLoCo and MuLoCo with and without Error Feedback (EF) accumulators.** We only report the memory required to store accumulators. `OuterOpt` is SGD with Nesterov Momentum.

Algorithm	`InnerOpt`	Mem. `InnerOpt`	Mem. `OuterOpt`
DiLoCo	AdamW	2× parameters	1× parameters
DiLoCo + EF	AdamW	3× parameters	1× parameters
MuLoCo	Muon	1× parameters	1× parameters
MuLoCo + EF	Muon	2× parameters	1× parameters

Recent works (Jordan et al., 2024; Liu et al., 2025; Shah et al., 2025) have demonstrated that Muon has the potential to replace AdamW as the de facto optimizer for pre-training LLMs. Given that Muon's orthogonalized updates are qualitatively different from those of AdamW, we hypothesize that, beyond its improved convergence, Muon may offer additional advantages in terms of compressibility. Given the potential of muon to be a powerful inner optimizer for DiLoCo and the absence of existing studies using compressed DiLoCo updates with error feedback, we set out to fill the gap. Specifically, we systematically compare variants of DiLoCo using different compression schemes (quantization, top-k sparsification, and more in the appendix) with and without error feedback terms and with different local optimizers: AdamW and Muon. Our contributions can be summarized as follows. When pre-training a 220M parameter transformer language model:

- We demonstrate error feedback consistently improves performance for top-k sparsified and quantized DiLoCo updates,
- We show that MuLoCo consistently converges faster than and to a lower loss than DiLoCo with and without compression, and
- We demonstrate that MuLoCo is more resilient to aggressive quantization than DiLoCo: when using 2-bit quantization and error feedback, MuLoCo reaches a lower loss than vanilla DiLoCo in the same number of steps while communicating $8\times$ less.

2. Background

DiLoCo. DiLoCo (Douillard et al., 2023) is a distributed optimization algorithm, which circumvents the need for communicating across all K workers when taking optimization steps. Instead, it takes H *local* gradient descent steps on each worker before communicating among them. The difference between DiLoCo and its earlier variants, federated averaging (McMahan et al., 2017) or Local SGD (Stich, 2018; Lin et al., 2018), is the choice of the inner and outer optimizer. While earlier works locally performed SGD and averaged parameters during communication, DiLoCo proposes to use AdamW (Loshchilov & Hutter, 2019) as the inner optimizer and SGD with Nesterov momentum as the outer optimizer. In the context of LLM pre-training, this has been shown 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., 2024). We outline the original DiLoCo algorithm in algorithm 1 of the appendix.

Compressed Local Updates. Existing works have explored many different compression schemes for neural network gradients and parameters. We will now introduce how schemes studied herein work when applied to a single matrix-valued parameters $\boldsymbol{W} \in \mathbb{R}^{m\times n}$:

- **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 $\boldsymbol{W}$.
- **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., 2023).

Muon (Jordan et al., 2024) is a newly proposed optimizer that has shown promising results outperforming AdamW for training MoEs and dense transformers (Liu et al., 2025; Shah et al., 2025). Muon maintains a single momentum accumulator, m, and computes its parameter update by first applying the newton-shulz algorithm (Björck & Bowie, 1971; Kovarik, 1970) to orthogonalize m before re-scaling by the learning rate. It has been shown that this procedure approximates UV^T, where $U\Sigma V^T = m$ is the Singular Value Decomposition of m (Bernstein, 2025). Since the communicated delta, Δ, in DiLoCo is a sum of subsequent inner-optimizer updates, the distinctly different update structure of Muon may result in improved or degraded performance under different compression schemes.

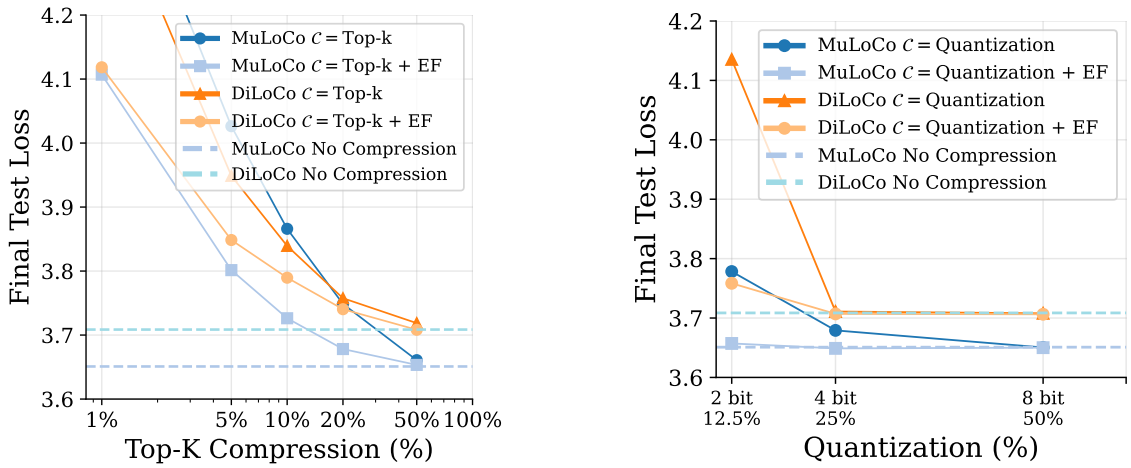


Figure 2: **Error feedback ablation for MuLoCo v.s. DiLoCo with compressed updates.** We vary the strength of the **(LEFT)** Top-k sparsification ($1 - 50\%$) and **(RIGHT)** Quantization (2,4, and 8 bits). EF designates runs using error feedback. The dashed lines report the final performance of the no-compression baseline (16-bit floats). We observe that EF consistently improves performance and that MuLoCo's advantage over DiLoCo improves as quantization increases.

3. Method

In this work, we make three methodological changes to the DiLoCo framework, which lead to communication savings over standard DiLoCo. The changes are highlighted with blue text in algorithm 1. Specifically, we change the inner optimizer (line 11), apply a compression function $\boldsymbol{C}$ to the communicated update (line 19), and experiment with using an error feedback EMA (lines 15-17).

DiLoCo algorithm We provide a detailed description of the DiLoCo algorithm and our modifications to it in Algorithm 1. The total number of outer steps is marked as N. At each outer step n, all workers will send the momentum of the compressed delta to the outer optimizer to update the model parameters using SGD with Nesterov Momentum. In the inner optimization phase, each worker $k \in [K]$ first samples data from their assigned shards, $\boldsymbol{D}_k$, and then utilizes the inner optimizer to perform local updates to the model for H steps. This significantly reduces the frequency of communication. After completing local optimization, each worker computes it delta, the difference between the model parameters before and after the local steps $\Delta_i = \boldsymbol{\theta}^{(t-H)} - \boldsymbol{\theta}_i^t$. We then apply error feedback and compression to Δ_i.

4. Empirical Evaluation

Our goal is to study, in the context of language model pretraining, (1) the performance of DiLoCo with Muon and AdamW inner optimizers, (2) the performance of error feedback accumulators for improving DiLoCo training with compressed updates, and (3) establish the effect of the inner optimizer on compression. To achieve this, we construct a systematic empirical study by borrowing the setup and most of the hyperparameter configuration from Scaling Laws for DiLoCo (Charles et al., 2025) at the 180M model size and ($K = 8$) workers. Leveraging this base setup, described in more detail below, with additional hyperparameter tuning when needed, we very the inner optimizer, compression mechanism, and ablate the use of error feedback.

Evaluation Task We select the relatively, small-scale 180M transformer language modeling task with $K = 8$ workers from (Charles et al., 2025) to make it tractable to run hyperparameter sweeps and a large-scale empirical evaluation. Following (Charles et al., 2025), we use a post-LN transformer with hidden dimension 1024, 12 hidden layers, and 16 attention heads; a global batch size of $B_G = 2^{18}$ tokens (see Figure 14 of (Charles et al., 2025)); $H = 30$ local steps; and a compute optimal 3.6B total tokens resulting in $13,733$ total steps and $T = 458$ communication steps with the setting of B_G and H. Unlike (Charles et al., 2025), we do not use KQ norm in our transformer; nor do we use Z-loss; we employ a slightly larger GPT-2 tokenizer meaning our model size is actually 220M parameters (but we keep the 3.6B total tokens); we use a sequence length of 512 to save memory; we decay the learning rate to $0.1\times$ the maximum value; and we pre-train on the FineWeb-EDU dataset (Lozhkov et al., 2024).

Hyperparameter tuning Due to the slight deviations from the setup of (Charles et al., 2025), all optimizers have their learning rates (η), inner learning rates (η_{in}), and error feedback beta (β) tuned on our full pre-training setup. The results of our hyperparameter search across DiLoCo and Data Parallel baselines is reported in the appendix 3. In summary, we sweep 8 learning rates for data-parallel training and over 16 for DiLoCo configurations. All hyperparameters are swept on a configuration with no compression, except for the error feedback beta, whose value is swept

using $C = 10\%$ Top-k compression + 4-bit quantization. Other optimizer-related hyperparameter choices are clarified below.

DiLoCo (`InnerOpt AdamW`) We use AdamW as a baseline inner optimizer for DiLoCo. We set the weight decay $\lambda = 7.2^{-5}$ for all experiments. Following (Charles et al., 2025) we set $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for all experiments.

MuLoCo (`InnerOpt Muon`) We use the Optax (DeepMind et al., 2020) implementation of Muon with a quintic Newton-Shulz iteration (Jordan et al., 2024). Muon is applied to hidden layers, while AdamW is used for the embeddings and output layers. We set the weight decay $\lambda = 7.2^{-5}$ and $\beta_1 = 0.9$ for both Muon and AdamW. We set AdamW's $\beta_2 = 0.99$. We use Nesterov momentum in Muon.

`OuterOpt` Following the optimal values found by (Charles et al., 2025) for AdamW DiLoCo in the the $K = 8$ workers 180M parameter task, we use SGD with Nesterov momentum ($\beta = 0.9$, $\eta = 0.8$) as the outer optimizer for all our DiLoCo and MuLoCo experiments.

All-Reduce Baselines For each inner-optimizer tested, we provide the equivalent all-reduce baseline which takes HT optimization steps with batch size B_{G}. This baseline is data-matched to the DiLoCo optimizers but requires H times more communication.

5. Results

The following section reports the results of our empirical evaluation. Section 5.1 compares DiLoCo to MuLoCo without compression and relative to data parallel baselines. Section 5.2 ablates the effect of error feedback when applying quantization, and Top-k sparsification to DiLoCo updates with Muon and AdamW inner optimizers.

5.1. MuLoCo vs DiLoCo: `InnerOpt` Ablation

In this section, we compare MuLoCo and DiLoCo to their respective Muon and AdamW data parallel baselines. All models use the same amount of data, but MuLoCo and DiLoCo communicate $H = 30$ times less. Figure 3 compares the final performance of the data parallel baselines (dashed lines) to the training curves of MuLoCo and DiLoCo. We observe that DiLoCo nearly matches the AdamW data parallel baseline, while MuLoCo outperforms it and matches its Muon baseline. In conclusion, without accounting for wall-clock time and at the scales tested, Muon is the clear best-performing local optimizer.

5.2. The Effect of Error Feedback accumulators when compressing MuLoCo updates

In this section, we ablate the performance of MuLoCo and DiLoCo when compression is applied during the communication step (see line 19 in Algorithm1). Specifically, we study top-k sparsification and quantization (see Sec. 2 for details of each). Our goal is to study how delta compression affects the different inner optimizers and establish effect of error feedback accumulators for each.

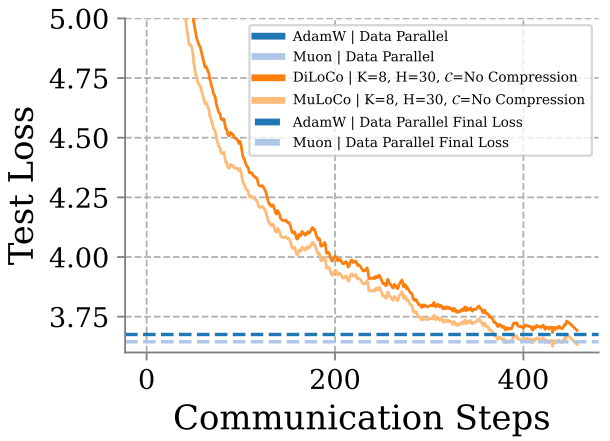


Figure 3: **Comparison to Data Parallel baselines.** We report the test loss at communication steps during training of MuLoCo and DiLoCo compared to their corresponding Data Parallel baselines (dashed lines). We observe that MuLoCo matches its data-parallel and that DiLoCo nearly matches its data-parallel baseline.

Figure 2 (a,b) reports top-k sparsification and quantization results for MuLoCo and DiLoCo with varying amounts of compression. We observe that across both subfigures all data points using error feedback (EF) accumulators outperform their counterparts without error feedback. Comparing between data points using error feedback, we observe that for all percentages of top-k compression MuLoCo consistently reaches a lower loss than DiLoCo. Tuning our attention to quantization + EF results, we observe that MuLoCo updates can be compressed to 2 bits with next to no performance degradation compared to the 16-bit baseline (dashed line), while DiLoCo suffers a performance degradation at 2 bits relative to its 16-bit baseline. Crucially, MuLoCo with 2-bit compression and EF has the same memory complexity as DiLoCo without compression, while communicating $8\times$ less. In summary, we have shown that error feedback is important for compressing DiLoCo updates and that using MuLoCo allows unprecedented quantization to 2 bits.

6. Conclusion

In conclusion, we have studied the performance of DiLoCo with Muon and AdamW inner optimizers under different compression schemes and demonstrated that MuLoCo (Muon inner optimizer) is a practical algorithm for low communication language model pre-training. Our experiments comparing different local optimizers under varying amounts of compression revealed that error feedback accumulators

are important for DiLoCo-style LLM pre-training with compressed deltas. We also found that when using Muon as the inner optimizer, the deltas can be compressed to 2 bits with next to no performance degradation. This means that MuLoCo with error feedback converges to a lower final loss and communicates $8\times$ less than standard AdamW DiLoCo while having the same memory complexity.

Acknowledgements

We acknowledge support from 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.*]. We also acknowledge resources provided by Compute Canada, Calcul Québec, and Mila.

References

Ahn, K. and Xu, B. Dion: A communication-efficient optimizer for large models. *arXiv preprint arXiv:2504.05295*, 2025. URL https://arxiv.org/abs/2504.05295.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

Basu, D., Data, D., Karakus, C., and Diggavi, S. Qsparse-local-sgd: Distributed sgd with quantization, sparsification, and local computations, 2019.

Bernstein, J. Deriving muon, 2025. URL https://jeremybernste.in/writing/deriving-muon.

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

Charles, Z., Teston, G., Dery, L., Rush, K., Fallen, N., Garrett, Z., Szlam, A., and Douillard, A. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco, March 2025. URL https://arxiv.org/abs/2503.09799.

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

Douillard, A., Feng, Q., Rusu, A. A., Chhaparia, R., Donchev, Y., Kuncoro, A., Ranzato, M., Szlam, A., and Shen, J. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Douillard, A., Donchev, Y., Rush, K., Kale, S., Charles, Z., Garrett, Z., Teston, G., Lacey, D., McIlroy, R., Shen, J., Ramé, A., Szlam, A., Ranzato, M., and Barham, P. Streaming diloco with overlapping communication: Towards a distributed free lunch, January 2025. URL https://arxiv.org/abs/2501.18512.

Jaghouar, S., Ong, J. M., and Hagemann, J. Opendiloco: An open-source framework for globally distributed low-communication training, July 2024. URL https://arxiv.org/abs/2407.07852.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://web.archive.org/web/20250122060345/https://kellerjordan.github.io/posts/muon/.

Joseph, C., Thérien, B., Moudgil, A., Knyazev, B., and Belilovsky, E. Can we learn communication-efficient optimizers? *CoRR*, abs/2312.02204, 2023.

Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 3252–3261, 2019. URL https://arxiv.org/abs/1901.09847.

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

Lin, T., Stich, S. U., and Jaggi, M. Don't use large mini-batches, use local SGD. *CoRR*, abs/1808.07217, 2018.

Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., Chen, Y., Zheng, H., Liu, Y., Liu, S., Yin, B., He, W., Zhu, H., Wang, Y., Wang, J., Dong, M., Zhang, Z., Kang, Y., Zhang, H., Xu, X., Zhang, Y., Wu, Y., Zhou, X., and Yang, Z. Muon is scalable for llm training, February 2025. URL https://arxiv.org/abs/2502.16982.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Lozhkov, A., Ben Allal, L., von Werra, L., and Wolf, T. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Peng, B., Quesnelle, J., and Kingma, D. P. Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. Adaptive federated optimization. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2003.00295.

Sani, L., Iacob, A., Cao, Z., Marino, B., Gao, Y., Paulik, T., Zhao, W., Shen, W. F., Aleksandrov, P., Qiu, X., and Lane, N. D. The future of large language model pre-training is federated. *arXiv preprint arXiv:2405.10853*, 2024. URL https://arxiv.org/abs/2405.10853.

Shah, I., Polloreno, A. M., Stratos, K., Monk, P., Chaluvaraju, A., Hojel, A., Ma, A., Thomas, A., Tanwer, A., Shah, D. J., Nguyen, K., Smith, K., Callahan, M., Pust, M., Parmar, M., Rushton, P., Mazarakis, P., Kapila, R., Srivastava, S., Singla, S., Romanski, T., Vanjani, Y., and Vaswani, A. Practical efficiency of muon for pretraining, 2025. URL https://arxiv.org/abs/2505.02222.

Shi, S., Chu, X., Cheung, K. C., and See, S. Understanding top-k sparsification in distributed deep learning. *CoRR*, abs/1911.08772, 2019.

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

Stich, S. U., Cordonnier, J., and Jaggi, M. Sparsified SGD with memory. *CoRR*, abs/1809.07599, 2018.

Vogels, T., Karimireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1905.13727.

Wang, J., Tantia, V., Ballas, N., and Rabbat, M. G. Slowmo: Improving communication-efficient distributed SGD with slow momentum. *CoRR*, abs/1910.00643, 2019.

Wang, J., Lu, Y., Yuan, B., Chen, B., Liang, P., De Sa, C., Re, C., and Zhang, C. CocktailSGD: Fine-tuning foundation models over 500Mbps networks. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 36058–36076. PMLR, 23–29 Jul 2023.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

Zhou, L., He, Y., Zhai, K., Liu, X., Liu, S., Ma, X., Ye, G., Jiang, Y.-G., and Chai, H. Fedcada: Adaptive client-side optimization for accelerated and stable federated learning. *arXiv preprint arXiv:2405.11811*, 2024. URL https://arxiv.org/abs/2405.11811.

A. Algorithm

Algorithm 1 DiLoCo with arbitrary `InnerOpt`, Nesterov Momentum `OuterOpt`, and compressed communication with optional error feedback. Additions beyond or potential modifications to the original DiLoCo algorithm made herein are colored blue.

Input:

N	Number of communication steps
K	Number of workers
H	Number of local steps
$\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$	Data shards
$\theta^{(0)}$	Initial model
`ERROR_FEEDBACK`	Error feedback flag
β	Error feedback coefficient
$\mathcal{C}$	Compression algorithm
μ	Outer momentum coefficient
η_{out}	Outer learning rate

$u^{(0)} = 0$ ▷ Initialize outer momentum accumulator
$\theta_i^{(1)} \leftarrow \theta^{(0)} \ \ \forall i$ ▷ Initialize all local parameter copies
$t = 0$
for outer step $n = 1 \ldots N$ **do**
  for worker $i = 1 \ldots K$ *in parallel* **do**
    for inner step $h = 1 \ldots H$ **do**
      $t = t + 1$;
      $x, y \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f_\theta(x, y; \theta_i^{(t)})$
      $g_i^{(n)} = \nabla_\theta \mathcal{L}$
      $\theta_i^{(t)} \leftarrow \texttt{InnerOpt}(\theta_i^{(t)}, \nabla\mathcal{L})$ ▷ Inner optimizer step
    $\Delta_i^{(t)} = \theta^{(t-H)} - \theta_i^t$ ▷ Compute parameter delta

    if `ERROR_FEEDBACK` **then**
      $\mathcal{E}_i^{(t)} \leftarrow \beta \mathcal{E}_i^{(t-H)} + \Delta_i^{(t)}$
      $\Delta_i^{(t)} = \mathcal{C}(\mathcal{E}_i^{(t)})$ ▷ Compress delta EMA
      $\mathcal{E}_i^{(t+1)} \leftarrow \mathcal{E}_i^{(t)} - \Delta_i^{(t)}$
    else
      $\Delta_i^{(t)} = \mathcal{C}(\Delta_i^{(t)})$ ▷ Compress parameter delta

    $\Delta^{(t)} \leftarrow \frac{1}{K} \sum_{i=1}^{K} \Delta_i^{(t)}$ ▷ All-reduce Outer Delta
    $u^{(t)} \leftarrow \mu u^{(t-H)} + \eta_{\text{out}} \Delta^{(t)}$ ▷ Update Nesterov accumulator
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \mu u^{(t)} - \eta_{\text{out}} \Delta^{(t)}$ ▷ Update Model Weights

B. Extended background

In extended experiments, we also compared with the following compression schemes:

1. **DCT Top-k** (Peng et al., 2024) introduces a novel compression mechanism for the low communication training of LLMs. Specifically, they first chunk the communicated matrix, $\boldsymbol{W}$, into $n/s + m/s$ chunks, c_i for $i \in (1, n/s + m/s)$, where $c_i \in \mathbb{R}^{s \times s}$ and s divides m and n. Then, they apply a Discrete Cosine Transform (DCT) to each chunk and apply top-k sparsification to its entries. Finally, the entries are all-gathered by each worker, decoded using the inverse transformation, and subsequently reassembled. Due to chunking, DCT Top-k sparsification allows using $\log_2(s) << \log_2(\max(m, n))$ bits to encode the top-k indices during communication.
2. **Random-k** sparsification involves keeping a random $k\%$ subset $\boldsymbol{W}$'s entries. The remaining entries are set to 0 and need to be communicated. This scheme has no additional communication overhead as the pattern can be recovered from the random seed.

C. Extended Empirical Results

We compare top-k to DCT top-k compression 4 and present extended pre-training curves in figures 5 and 5. When comparing top-k to DCT top-k sparsification, we observe that DiLoCo seems to benefit more than MuLoCo. At 10% and 20% both methods show a small improvement while at 5% top-k is better than its DCT variant for MuLoCo but not for DiLoCo which still benefits from DCT. At 1% sparsification, Top-k is strictly better than its DCT variant.

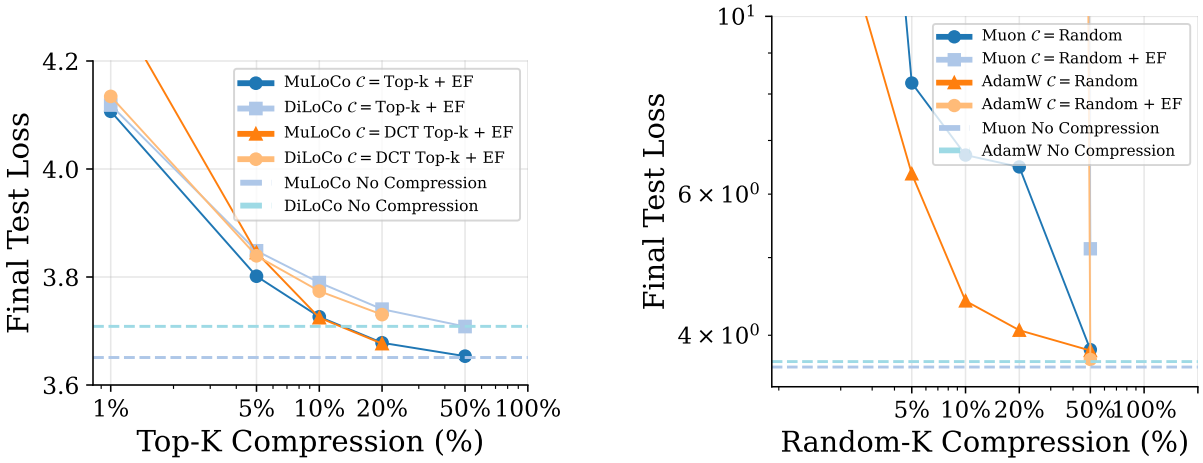
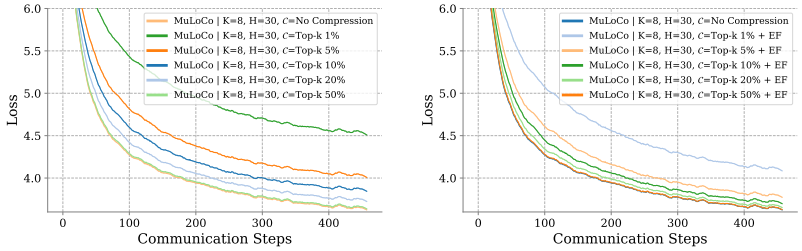
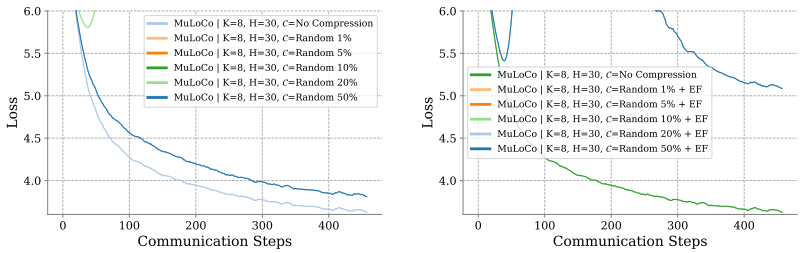


Figure 4: **Top-k v.s. DCT Top-k sparsification and random sparsification results.** We use DCT compression with a chunk size $s = 128$. We observe that DCT compression slightly improves performance for both MuLoCo and DiLoCo at 10% and 20% compared to Top-k. For 5% MuLoCo's performance degrades while DiLoCo still works well. We observe that Random-k compression results lead to fast performance deterioration.



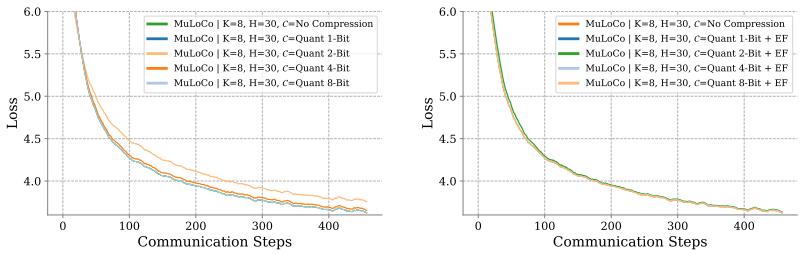
(a) Muon Top-K

(b) Muon Top-K (EF)



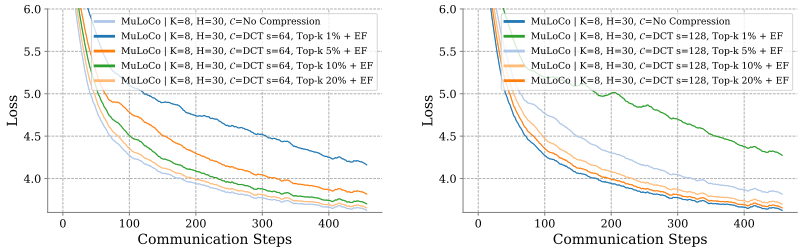
(c) Muon Random-K

(d) Muon Random-K (EF)



(e) Muon Quantized

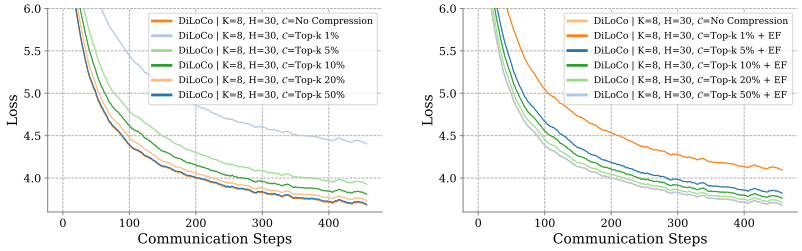
(f) Muon Quantized (EF)



(g) Muon DeMo s=64 (EF)

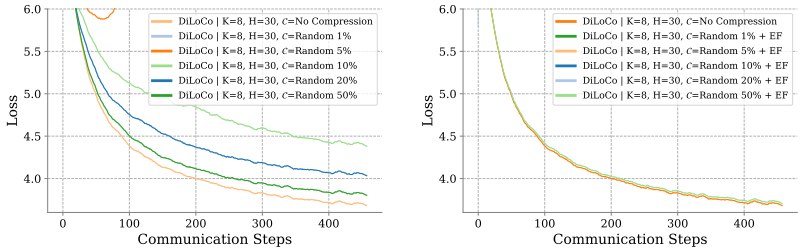
(h) Muon DeMo s=128 (EF)

Figure 5: **Training curves for Muon DiLoCo with different training curves.** We report the training loss for variants of DiLoCo with different inner optimizers. All hyperparameters were tuned on a setup without any compression, except the error feedback β parameter. Any non-visible curves that appear in the legend either overlap with another curve or do not reach low enough loss to be seen.



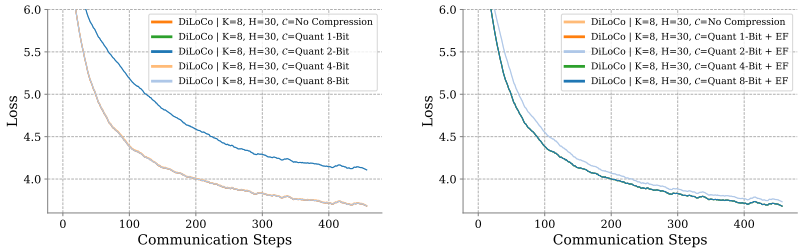
(a) Muon Top-K

(b) Muon Top-K (EF)



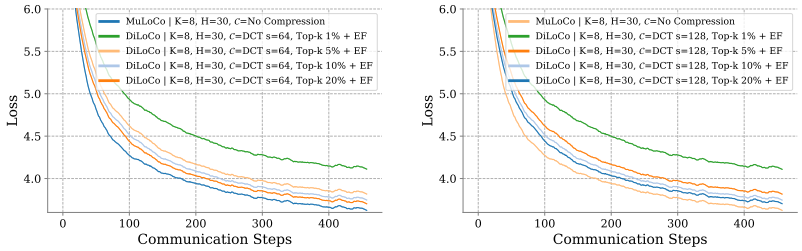
(c) Muon Random-K

(d) Muon Random-K (EF)



(e) Muon Quantized

(f) Muon Quantized (EF)



(g) Muon DeMo s=64 (EF)

(h) Muon DeMo s=128 (EF)

Figure 6: **Training curves for AdamW DiLoCo with different training curves.** We report the training loss for variants of DiLoCo with different inner optimizers. All hyperparameters were tuned on a setup without any compression, except the error feedback β parameter. Any non-visible curves that appear in the legend either overlap with another curve or do not reach low enough loss to be seen.

D. Extended Litterature Review

In the following section, we review related literature on communication-efficient distributed learning across two main categories (Sec. D.1, D.2).

D.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). The algorithm has since been shown theoretically and empirically to lead to communication savings (Stich, 2018; Lin et al., 2018). Follow-up works have enhanced the algorithm by using more sophisticated server-side optimizers (Wang et al., 2019; Joseph et al., 2023; Reddi et al., 2021). Most recently, works have also studied adaptive client-side otpimizers (Zhou et al., 2024; Douillard et al., 2023; Sani et al., 2024). Notably, in the context of i.i.d. language modeling, DiLoCo (Douillard et al., 2023) uses SGD and Nesterov momentum as the server-side optimizer and AdamW on local workers. In follow-up works, DiLoCo has shown the method scales to much larger workloads and allows for more parallelism than data parallel training (Jaghouar et al., 2024; Charles et al., 2025).

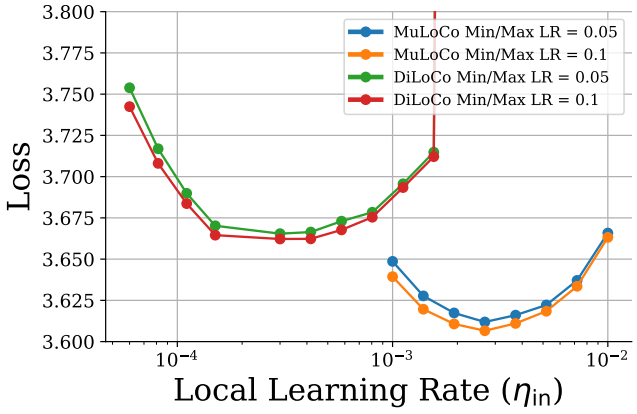
D.2. Reducing the communication of optimization algorithms

Another approach to reducing communication overhead in distributed training is to reduce the number of bits communicated. This is typically done by sparsifying (Stich et al., 2018; Shi et al., 2019), quantizing (Alistarh et al., 2017), or computing low-rank approximations of the gradient (Zhao et al., 2024; Vogels et al., 2019). In CocktailSGD, (Wang et al., 2023) combine sparsification, quantization, and error feedback (Karimireddy et al., 2019) to achieve substantial communication reduction for fine-tuning LLMs. In the context of communicate-every-step LLM pre-training, two recent works also apply compression schemes with error feedback and only all-reduce compressed quantities to lower communication costs (Peng et al., 2024; Ahn & Xu, 2025). In one of the most closely related works to ours, (Basu et al., 2019) apply top-k sparsification and quantization to error-corrected moving averages of parameter deltas in the context of local SGD for ResNets. In contrast, we study this in the context of transformer language models for many more local steps and with multiple different local optimizers (Muon and AdamW). Streaming DiLoCo (Douillard et al., 2025) is the most closely related work to our own. Both works share the same goal: to reduce the communication overhead of DiLoCo. The differences are that Streaming DiLoCo only evaluates a single inner optimizer, AdamW, and they do not consider combining top-k sparsification and quantization with error feedback.

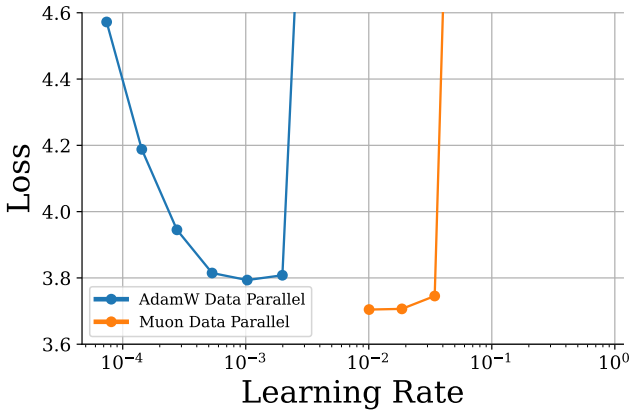
E. Hyperparameter Selection Details

Table 2: **Best hyperparameters found for each optimizer.** No value reported means that the hyperparameter was not swept.

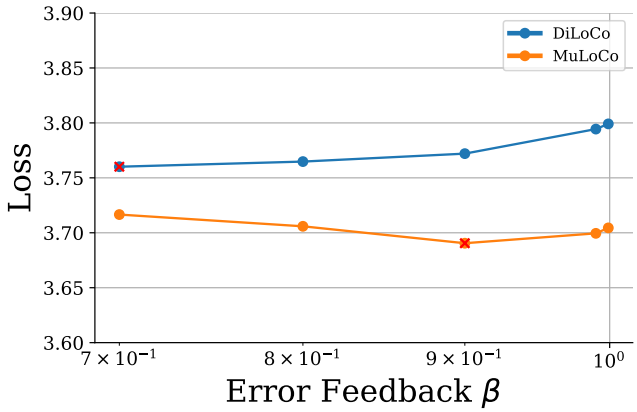
Optimizer	Peak LR	Min/Max LR Ratio	Error Feedback β
DiLoCo	0.0003	0.1	0.7
MuLoCo	0.0026827	0.1	0.9
AdamW Data Parallel	0.0001	–	–
Muon Data Parallel	0.01	–	–



(a) DiLoCo



(b) DataParallel



(c) Error Feedback

Figure 7: **Hyperparameter sweeps.** We tune Min/Max LR ratio, maximum η_{in} of DiLoCo and MuLoCo, the learning rate of the data parallel baselines, We report the final training loss on the y-axis and the log-hyperparameter on the x-axis. Low communication optimizers uses $K = 8$ workers and $H = 30$ inner steps. The global batch size is always $B_{\text{loc}} = 262k$ tokens. The error correction β is tuned on a setup whose compression algorithm combines top-k and quantization to be representative of strong but realistic compression.

Table 3: **Hyperparameter sweep values for DiLoCo, MuLoCo, and Data Parallel Baselines.** We swept more values for DiLoCo than MuLoCo since our initial sweep did not show a clear minimum value.

Hyperparameter	#	Values
DiLoCo		
η_{in}	12	$\{6 \times 10^{-5},\ 8.14 \times 10^{-5},\ 1.11 \times 10^{-4},\ 1.5 \times 10^{-4},\ 3 \times 10^{-4},\ 4.17 \times 10^{-4},\ 5.79 \times 10^{-4},\ 8.05 \times 10^{-4},\ 1.12 \times 10^{-3},\ 1.55 \times 10^{-3},\ 2.16 \times 10^{-3},\ 3 \times 10^{-3}\}$
Min/Max LR ratio	2	$\{0.05,\ 0.1\}$
MuLoCo		
η_{in}	8	$\{0.001,\ 1.39 \times 10^{-3},\ 1.93 \times 10^{-3},\ 2.68 \times 10^{-3},\ 3.73 \times 10^{-3},\ 5.18 \times 10^{-3},\ 7.20 \times 10^{-3},\ 0.01\}$
Min/Max LR ratio	2	$\{0.05,\ 0.1\}$
Data Parallel (AdamW)		
Learning Rate	8	$\{7.4 \times 10^{-5},\ 1.43 \times 10^{-4},\ 2.76 \times 10^{-4},\ 5.33 \times 10^{-4},\ 1.03 \times 10^{-3},\ 1.99 \times 10^{-3},\ 3.83 \times 10^{-3},\ 7.4 \times 10^{-3}\}$
Data Parallel (Muon)		
Learning Rate	8	$\{0.0101,\ 0.0186,\ 0.0343,\ 0.0635,\ 0.1173,\ 0.2167,\ 0.4005,\ 0.74\}$
Error Feedback		
β	5	$\{0.7,\ 0.8,\ 0.9,\ 0.99,\ 0.999\}$

Table 4: **DiLoCo Optimizers Hyperparameters.**

Description	Value
Shared	
Total Local Iterations (NH)	$13,740$
Total Communication Steps (N)	458
Number of Local Steps (H)	30
Number of Workers (K)	8
Linear Warmup iters (T_{warmup})	1000
Sequence Length	512
Local Batch Size B_{loc}	64
Gradient clipping	1.0
Decay	Cosine
AdamW DiLoCo	
Max learning rate (η_{max})	$3 \cdot 10^{-4}$
Min learning rate (η_{min})	$3 \cdot 10^{-5}$
Muon DiLoCo	
Max learning rate (η_{max})	$2.6827 \cdot 10^{-3}$
Min learning rate (η_{min})	$2.6827 \cdot 10^{-4}$
AdamW Data Parallel	
Max learning rate (η_{max})	$1 \cdot 10^{-4}$
Min learning rate (η_{min})	$1 \cdot 10^{-5}$
Muon Data Parallel	
Max learning rate (η_{max})	$1 \cdot 10^{-2}$
Min learning rate (η_{min})	$1 \cdot 10^{-3}$

Table 5: **Transformer Language Model Hyperparameters.**

Description	Value
Dense Transformer	
Parameters	$228,949,073$
Embedding Parameters	$51,463,168$
Weight tying	False
Num attention heads	16
Num layers	12
Hidden size	1024
FFN Hidden size	4096
FFN Activation	GeLU
Positional embedding	Learned
Vocab Size	50257