Communication-Efficient Language Model Training Scales Reliably and Robustly: Scaling Laws for DiLoCo

Zachary Charles
Google Research
zachcharles@google.com

Gabriel Teston
Google Search
teston@google.com

Lucio Dery
Google DeepMind
ldery@google.com

Keith Rush
Google DeepMind
krush@google.com

Nova Fallen
Google Research
nfallen@google.com

Zachary Garrett
Google Research
zachgarrett@google.com

Arthur Szlam
Google DeepMind
aszlam@google.com

Arthur Douillard
Google DeepMind
douillard@google.com

Abstract

As we scale to more massive machine learning models, the frequent synchronization demands inherent in data-parallel approaches create significant slowdowns, posing a critical challenge to further scaling. Recent work [11, 24] develops and analyzes an approach (DiLoCo) that relaxes synchronization demands via periodic synchronization. However, these works do not carefully analyze how DiLoCo's behavior changes with model size. In this work, we study the scaling law behavior of DiLoCo when training LLMs under a fixed compute budget. We focus on how algorithmic factors, including number of model replicas, hyperparameters, and token budget affect training in ways that can be accurately predicted via scaling laws. We find that DiLoCo scales both predictably and robustly with model size. When well-tuned, DiLoCo scales better than data-parallel training with model size, and can outperform data-parallel training even at small model sizes. Our results showcase a more general set of benefits of DiLoCo than previously documented, including increased optimal batch sizes, improved downstream generalization with scale, and improved evaluation loss for a fixed token budget.

1 Introduction

Large language models (LLMs) are typically trained via large-batch distributed data-parallel methods. However, bandwidth and communication constraints can become bottlenecks at larger scales due to frequent synchronizations, posing a critical challenge to further scaling. As a remedy, Douillard et al. [11] propose DiLoCo (*Distributed Low-Communication*), a generalization of algorithms like Local SGD [34, 54] and FedAvg [36], which enables training of LLMs in parallel across "islands" of compute (such as datacenters connected via low-bandwidth networks) by performing parallel training of models with only periodic synchronization. While empirically successful, DiLoCo exists within a broader context of work on communication-efficient LLM training methods [58, 43, 56, 3, 24, 30]. Due to the breadth of related work, we defer a more thorough overview to Section A.

Unlike communication-reduction methods such as quantization and sparsification, DiLoCo fundamentally alters training dynamics [49]. While Douillard et al. [11] and Jaghouar et al. [24] show

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

that DiLoCo yields comparable evaluation metrics to data-parallel training at moderate model scales, it is unclear how data-parallel training and DiLoCo compare at larger model scales. Moreover, DiLoCo has extra hyperparameters not present in data-parallel training that may be computationally prohibitive to tune at large enough scales. This points to the need for DiLoCo *scaling laws*. We focus on two specific scaling laws: (1) predictions for evaluation loss as a function of model size and (2) predictions for optimal hyperparameter choices for a given model size (which can obviate the need to perform expensive hyperparameter tuning). In both cases, we are explicitly interested in how these compare to analogous scaling laws for data-parallel training.

Throughout we consider the task of pre-training a model of size N on D tokens. We are concerned with predicting, for both data-parallel training and DiLoCo training, and as a function of N: (1) the evaluation loss L after training, computed on a held-out set, and (2) optimal hyperparameter settings. One path towards scaling laws for DiLoCo would be to view them as modifications of scaling laws for data-parallel training [26, 20]. However, the facets of DiLoCo that are key to its communication-efficiency also make such an approach infeasible. First, DiLoCo operates by training M models in parallel, with periodic synchronization every H steps. The values of M and H depend on the ecosystem of compute available (such as bandwidth across datacenters), and are absent in scaling laws for data-parallel training. Second, DiLoCo uses a bi-level optimization framework; each model replica performs data-parallel training, but upon synchronization we apply an "outer" optimization step [11]. This means that DiLoCo has "outer" hyperparameters not present in Data-Parallel training that cannot be inferred from data-parallel hyperparameter scaling laws.

Contributions. We develop scaling laws for data-parallel training and DiLoCo from the ground up. Fixing the number of tokens D to be the "Chinchilla-optimal" number of tokens [20], we model the evaluation loss and optimal hyperparameters as functions of model size N and (for DiLoCo) the number of replicas M. [1] We empirically estimate these functions using the final evaluation loss attained by models trained with both algorithms for varying hyperparameters (including learning rate, batch size, and "outer" learning rate for DiLoCo), model sizes (varying N over 9 model sizes ranging from 35 million to 2.4 billion parameters), and numbers of DiLoCo replicas M.

Our scaling laws predict that in many settings, the more communication-efficient DiLoCo algorithm actually yields better evaluation loss than data-parallel training for the same token budget. Utilizing our scaling laws to predict the hyperparameters for DiLoCo, we tested these predictions when training models with 4 billion and 10 billion parameters. The scaling laws proved accurate, with DiLoCo with $M = 2$ replicas outperforming data-parallel training as predicted, even while reducing total communication by a factor of over 100.

We show that DiLoCo incurs a variety of benefits in comparison to data-parallel training, including (1) increased optimal batch size, allowing for greater horizontal scalability, (2) greater reductions in evaluation loss as model size increases, and (3) significantly less wall-clock training time. One potentially surprising finding: DiLoCo improves training even when communication is not a bottleneck. DiLoCo with $M = 1$ (an enhanced version of the Lookahead optimizer [62]) does not reduce communication but achieves lower evaluation loss at all model scales. It is also more robust to larger batch sizes, greatly reducing wall-clock training time.

2 Preliminaries

Table 1: General Notation

Symbol	Meaning
θ	Model weights
N	Model size
L	Evaluation loss
T	Training steps
D	Token budget
C	Total FLOPs

Table 2: Algorithm-Specific Notation

Symbol	Data-Parallel	DiLoCo
γ	Learning rate	Inner learning rate
η	–	Outer learning rate
B	Batch size	Global batch size
M	–	DiLoCo replicas
H	–	Synchronization cadence

[1] While we fix $H = 30$ for these scaling laws, we provide extensive ablations on the role of H in Section 5.

Throughout, we let θ denote the model parameters. We let $\theta^{(t)}$ denote the model at step t. Since DiLoCo operates on M parallel model parameters, we will use subscript notation θ_m to denote the m-th model. When there is no subscript, the parameters are assumed to be replicated across all DiLoCo replicas. For a batch of data x, we let $f(\theta, x)$ denote the loss of θ on the batch of data.

DiLoCo [11] is a technique designed for training models in the presence of communication constraints. It is motivated by the training of large models across devices that are not all connected by low-latency bandwidth. To avoid incurring latency costs, DiLoCo trains M models in parallel (ideally, training each one with co-located compute connected via low latency bandwidth), only synchronizing the models every H steps. This is similar to the FedOpt algorithm used in federated learning [47], but with the important difference that the replicas maintain their inner optimizer state across rounds.

DiLoCo applies a bi-level optimization framework across multiple models: each DiLoCo replica has its own model $\theta_m^{(t)}$, and there is a global model $\theta^{(t)}$. At every step, each replica takes an *inner* optimization step (`InnerOpt`). Every H steps, each replica computes the $\Delta_m^{(t)} = \theta^{(t-H)} - \theta_m^{(t)}$, the difference between the replica's current model and the most recent global model. We average these differences across replicas, resulting in $\Delta^{(t)}$ which we refer to as an *outer gradient*. We treat this as a gradient estimate of the outer model[2] and an outer optimization step (`OuterOpt`) to the outer model $\theta^{(t-H)}$. This yields an updated outer model $\theta^{(t)}$ which is broadcast to all replicas and set as their current inner model. We give full pseudo-code for DiLoCo in Algorithm 1.

Throughout, we perform model training via distributed data-parallel training (Data-Parallel), and DiLoCo. In Data-Parallel, at each step we distribute a batch of B tokens across workers. We then compute a batch gradient and perform optimization with a learning rate of γ. In DiLoCo, at each step t, we take a global batch of tokens of size B consisting of B/S sequences each of length S. We partition the B/S sequences across the M DiLoCo replicas, so each replica receives B/SM sequences, each of length S. Thus, the *global* token batch size is B, but each DiLoCo replica uses a local token batch size B/M. As in Data-Parallel, each replica computes a batch gradient and applies an inner optimization step with learning rate of γ. Unlike Data-Parallel, DiLoCo does outer optimization (on outer-gradients computed in parameter space) every H steps with learning rate η.

Algorithm 1 DiLoCo

Require: Loss function $f(\theta, x)$, batch size B, number of replicas M, synchronization cadence H, initial model weights $\theta^{(0)}$, data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$
Require: Optimizers `InnerOpt` and `OuterOpt`
$\forall m, \theta_m^{(0)} \leftarrow \theta^{(0)}$
for `step` $t = 1 \ldots T$ **do**
  parallel for `replica` $m = 1 \ldots M$ **do**
    Receive a batch $x_m^{(t)} \sim \mathcal{D}_m$ of size B/M
    $g_m^{(t)} \leftarrow \nabla_\theta f(\theta_m^{(t-1)}, x_m^{(t)})$
    $\theta_m^{(t)} \leftarrow \texttt{InnerOpt}(\theta_m^{(t-1)}, g_m^{(t)})$
  end parallel for
  if $t \bmod H = 0$ **then**
    $\Delta_m^{(t)} \leftarrow \theta^{(t-H)} - \theta_m^{(t)}$
    $\Delta^{(t)} \leftarrow \frac{1}{M} \sum_{m=1}^{M} \Delta_m^{(t)}$
    $\theta^{(t)} \leftarrow \texttt{OuterOpt}(\theta_m^{(t-H)}, \Delta^{(t)})$
    $\forall m, \theta_m^{(t)} \leftarrow \theta^{(t)}$

When comparing Data-Parallel and DiLoCo, we use the same model size N and token budget D. When computing evaluation loss L on some held-out set, for Data-Parallel we use the current model, and for DiLoCo we use the most recent global model. We summarize algorithm-independent notation in Table 1 and algorithm-specific notation in Table 2. An important comparison is Data-Parallel versus DiLoCo with $M = 1$. While similar, they are not identical as DiLoCo with $M = 1$ uses an outer optimizer step with optimizer `OuterOpt`, which is often set to SGD with Nesterov momentum [11].[3]

3 Experimental Methodology

Model architecture. We use a Chinchilla-style decoder-only transformer [20]. As suggested by Wortsman et al. [60] and Jaghouar et al. [24], we use QK-LayerNorm to reduce sensitivity to learning

[2]Note that $\Delta^{(t)}$ is generally not a gradient of any function, as it can evince non-conservative dynamics [5].

[3]This is similar in spirit to the fast- and slow-momentum steps in AdEMAMix [41], but yields different training dynamics since it uses a gradient estimate computed by linearizing across multiple training steps.

Table 3: Model details, including size, number of layers, layer dimensions, and token budgets. For the larger models (4B and 10B) we use scaling laws to predict optimal hyperparameters, rather than performing extensive hyperparameter tuning.

Model Scale	Transformer Layers	Attention Heads	QKV Dimension	Hidden Dimension	Token Budget	Hyperparameter Sweep
35M	6	8	512	2,048	70M	✓
90M	9	12	768	3,072	1.8B	✓
180M	12	16	1,024	4,096	3.6B	✓
330M	15	20	1,280	5,120	6.6B	✓
550M	18	24	1,536	6,144	11B	✓
1.3B	24	32	2,048	8,192	26B	✓
2.4B	30	40	2,560	10,240	48B	✓
4B	36	48	3,072	12,288	80B	✗
10B	48	64	4,096	16,384	200B	✗

rate. We also use z-loss regularization [8]. We use a vocabulary size of 32,768: 32,000 in-vocabulary words, and extra tokens for BOS and out-of-vocabulary. We pack multiple sequences into each batch, with a sequence length of 2,048 throughout. We pre-train a family of models, varying the number of transformer layers, number of attention heads, QKV dimension, and feed-forward layer hidden dimension (see Table 3). We use the Chinchilla-optimal token budget [20] unless otherwise noted. We do extensive hyperparameter sweeps on all models except the two largest (4B and 10B).

Datasets. Unless otherwise noted, we use the train split of the C4 dataset for training [45]. We report evaluation metrics on C4's held-out validation set. We compute downstream zero-shot evaluation metrics on 3 tasks: HellaSwag [61], Piqa [4], and Arc-Easy [9]. In overtraining ablations (Section 5), we use the Dolma dataset [53] instead of C4, to avoid doing more than a single epoch of training.

Optimizers. We use AdamW [32] as the optimizer for Data-Parallel and inner optimizer for DiLoCo, with $\beta_1 = 0.9$, $\beta_2 = 0.99$. Following [59], we use a weight decay parameter of $\lambda = T^{-1}$ where T is the number of training *steps*. We do 1000 steps of warmup followed by cosine learning rate decay to 5% of the peak learning rate. We clip (inner) gradients to a norm of 1. We do not clip outer gradients. For DiLoCo, we use SGD with Nesterov momentum [55] as the outer optimizer, using a momentum of 0.9 and constant outer learning rate. Unless otherwise specified, we set $H = 30$.

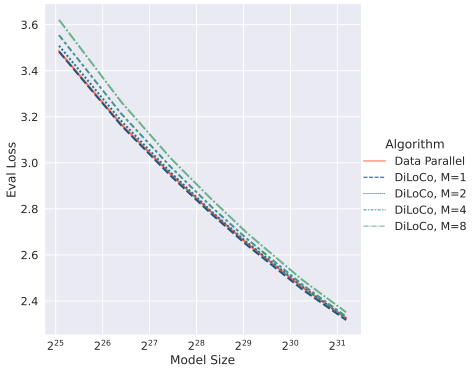
Implementation. We use a modified version of NanoDO [31] that uses DrJAX [49] to parallelize training steps across replicas. We use bfloat16 representation of model weights and gradients.

Idealized wall-clock time. For each experiment, we compute an idealized end-to-end wall-clock time for training. Our model assumes that we are training a model across multiple datacenters. Within a datacenter, we have a high-bandwidth network. Across datacenters, we have a high-, medium-, or low-bandwidth network. For details on the idealized wall-clock time, see Section B.

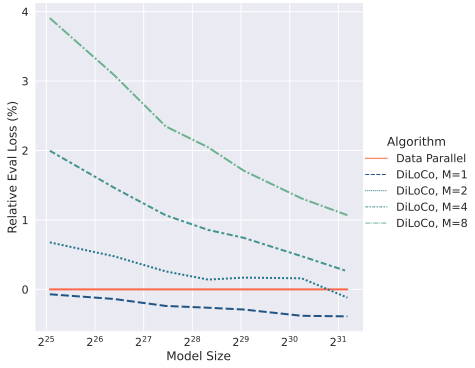
Scaling law experiments. We perform comprehensive hyperparameter sweeps for Data-Parallel and DiLoCo on models ranging from 35M to 2.4B We sweep over the learning rate γ using integer powers of $\sqrt{2}$ and batch size B using powers of 2. For DiLoCo, we train using $M = 1, 2, 4, 8$ and sweep the outer learning rate η over $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. We sweep (inner) learning rate and batch size as needed until the minimum loss value is obtained on an interior point. Using this data, we derive scaling laws to predict evaluation loss and optimal hyperparameters for larger models. We use the predicted hyperparameters to train models with 4B and 10B parameters, in order to validate the scaling laws empirically.

4 Empirical Findings

Before discussing our process for fitting the specific scaling laws, we talk about the empirical results and four critical findings that are worth highlighting independently.



(a) Evaluation loss for various algorithms, as a function of model size N.



(b) Percentage difference in evaluation loss, relative to Data-Parallel.

Figure 1: We compare Data-Parallel to DiLoCo for varying model sizes N. For all M, DiLoCo improves monotonically with respect to Data-Parallel as N increases.

Finding 1: DiLoCo's evaluation loss improves relative to Data-Parallel as N increases. We give the evaluation loss achieved by Data-Parallel and DiLoCo for each model size N, along with percentage differences relative to Data-Parallel, in Figure 1. We see that for all values of M, the percentage difference strictly decreases with N. At N = 2.4B, DiLoCo with $M = 1$ or 2 performs better than Data-Parallel. We validate this by training 4B and 10B models with hyperparameters set via our scaling laws. The results are in Table 4. We see that at 4B and 10B scales, DiLoCo with $M = 1, 2$ continue to do better than Data-Parallel. We discuss the scaling laws in detail in Section 6.

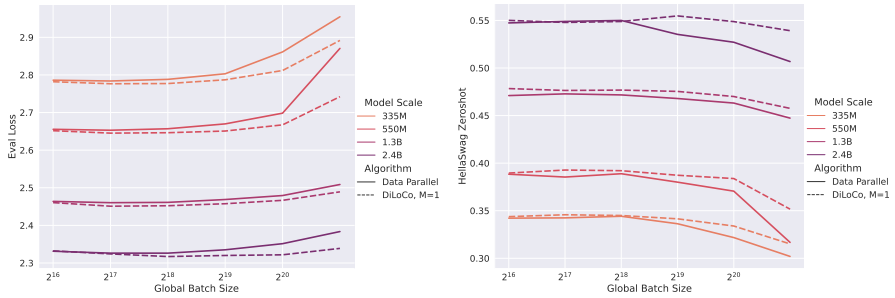
Table 4: Evaluation results on 4B and 10B models, using hyperparameters predicted by scaling laws. We indicate settings where DiLoCo reaches lower loss than Data-Parallel in bold.

Algorithm	Loss	
	4B	10B
Data-Parallel	2.224	2.090
DiLoCo, $M = 1$	**2.219 (-0.22%)**	**2.086 (-0.19%)**
DiLoCo, $M = 2$	**2.220 (-0.18%)**	**2.086 (-0.19%)**
DiLoCo, $M = 4$	2.230 (+0.18%)	2.096 (+0.29%)

Finding 2: DiLoCo with $M = 1$ attains lower evaluation loss than Data-Parallel across model scales. DiLoCo with $M = 1$ also achieves higher downstream zero-shot accuracy, as we show in Figure 2. These plots also show that DiLoCo with $M = 1$ also exhibits greater stability with respect to batch size; doubling or quadrupling the batch size greatly reduced performance of Data-Parallel, but had little effect on DiLoCo, $M = 1$, as depicted in Figure 2.

Finding 3: DiLoCo increases optimal batch size. We plot evaluation loss as a function of batch size in Figure 3. Optimal batch size increases when using DiLoCo, and subsequently increases with M. As batch size increases, Data-Parallel becomes worse than DiLoCo with $M = 2, 4$, and eventually, $M = 8$. We show that the same occurs for downstream tasks in the appendix (see Figure 10). This means that DiLoCo exhibits more *horizontal scalability*, reducing training time via resource parallelization in addition to reducing communication. To show this, we plot an idealized wall-clock time when training under networks of varying bandwidth in Figure 4. DiLoCo's tolerance for larger batch sizes allows it to achieve comparable loss to Data-Parallel significantly faster.

Finding 4: Optimal outer learning rate is constant with N. While optimal inner learning rate varies with model size N, the optimal outer learning rate η for DiLoCo is independent of N and depends only on M. As shown in Figure 5, for sufficiently large models ($N \geq$ 335M), the best η for each M is constant. Larger values of M seem to necessitate larger η. This is consistent with



(a) Evaluation loss.

(b) Zero-shot accuracy on HellaSwag.

Figure 2: Evaluation loss and downstream accuracy of Data-Parallel and DiLoCo with $M = 1$ for varying model and global batch sizes (measured in tokens).

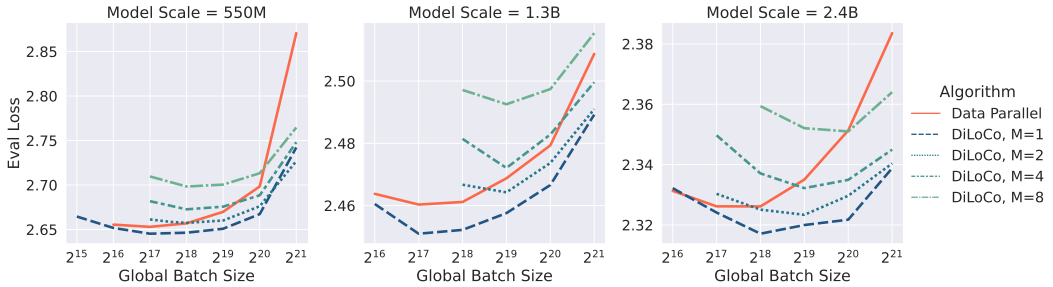
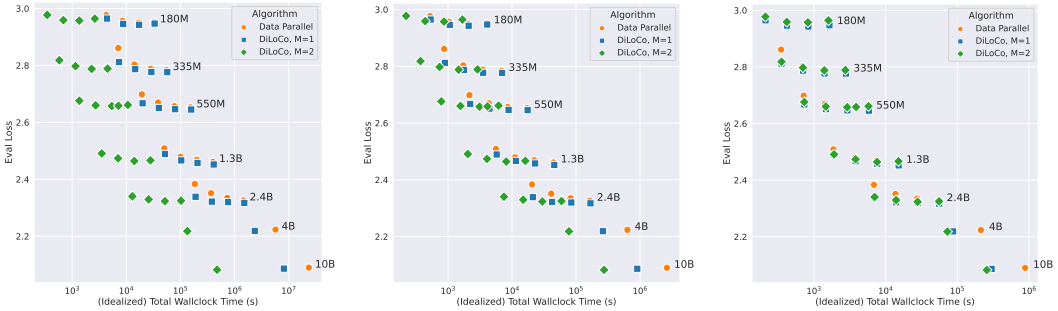


Figure 3: Evaluation loss of Data-Parallel and DiLoCo as a function of global batch size (in tokens). We see similar results for other model sizes, but omit for conciseness.

prior findings that outer learning rate should increase as a function of number of clients in federated learning settings [6].

5 Ablations

Synchronization cadence. Our experiments above all use a synchronization cadence H of 30. We perform an ablation over H for varying M and model sizes N. For each setting, we take the optimal inner learning rate and global batch size from above, and sweep H over $\{1, 5, 10, 30, 100, 300\}$ and



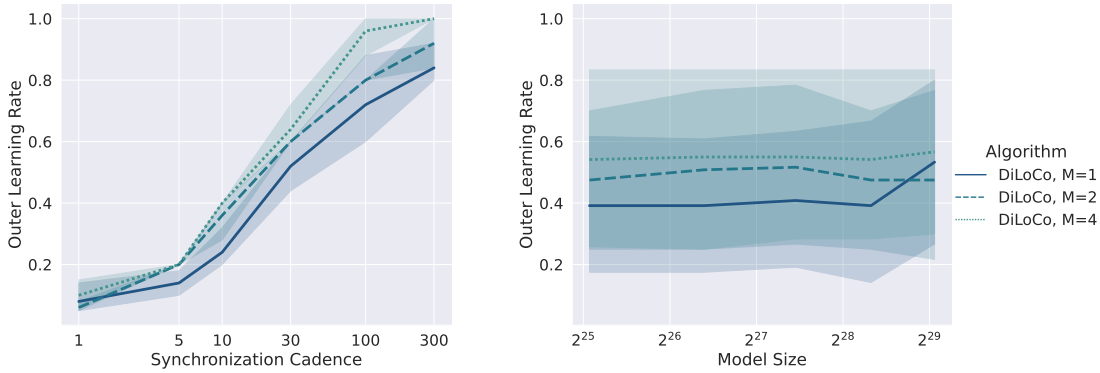
(a) Network with a bandwidth of 10 gigabits/s and a latency of 10^{-2} seconds (**low-bandwidth**).

(b) Network with a bandwidth of 100 gigabits/s and a latency of 10^{-3} seconds (**medium-bandwidth**).

(c) Network with a bandwidth of 400 gigabits/s and a latency of 10^{-4} seconds (**high-bandwidth**).

Figure 4: Idealized wall-clock time (see Section B) when training with Data-Parallel and DiLoCo across compute nodes connected via high-, medium-, and low-bandwidth networks, for varying model sizes. For models up to 2.4B, we also vary global batch size. For 4B and 10B models, we use the batch size predicted by scaling laws (see Section 6).

Figure 5: The best outer learning rate for DiLoCo for varying M and model size N. We select the best outer learning rate over $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, optimizing over inner learning rate γ and global batch size B. For sufficiently large models, the best outer learning rate is constant.



(a) Optimal outer learning rate for each synchronization cadence. Shaded regions represent the variance across model sizes.

(b) Optimal outer learning rate for each model size. Shaded regions represent the variance across synchronization cadences.

Figure 6: Optimal outer learning rate η as a function of synchronization cadence H and model size N. The optimal η increases with H and M (left), but is independent of model size N (right).

η over $\{0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$. We first study whether the observation in Section 4, that η should be tuned independently of N, holds for other values of H. We give results in the affirmative in Figure 6. We see that across model scales, the optimal learning rate is essentially only a function of the number of replicas M and the synchronization cadence H, and is essentially independent of model size N. There is some slight variation, though this is likely due to not re-tuning the inner learning rate. Moreover, our results actually show a potentially counter-intuitive phenomenon: The optimal outer learning rate *increases* with H, even though the outer gradients increase in size as H increases. We present additional analyses of synchronization cadence, including how H impacts evaluation loss and compute utilization, in Section E.1.

Overtraining. Above, we used the Chinchilla-optimal amount of tokens for each model size [20]. It is often beneficial to perform *overtraining*, using more tokens than that [17]. We conduct ablations on various *overtraining multipliers*. Given an overtraining multiplier $\zeta \geq 1$, we train on $D = 20N\zeta$ tokens, so that $\zeta = 1$ corresponds to the Chinchilla-optimal number of tokens. Our results are in Figure 7. Qualitatively, the scaling remains essentially unchanged as we do more overtraining. We did not re-tune any hyperparameter in these experiments. For each model size and algorithm, we simply took the best-performing hyperparameters from our Chinchilla-optimal experiments on C4. This means that the consistency of DiLoCo as we overtrain held despite the fact that for $M > 1$, we used larger batch sizes than Data-Parallel. We shows this in detail in Section E.2.

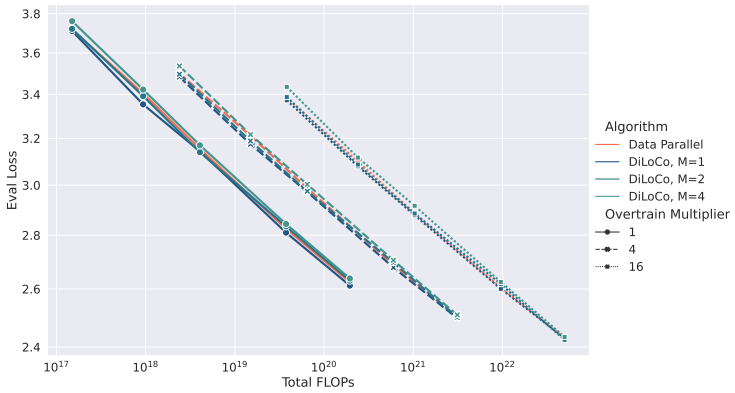


Figure 7: Evaluation loss as a function of FLOPs, for varying algorithms and amounts of over-training. For each overtrain multiplier, the curves are all essentially parallel lines.

6 Scaling Laws

We now discuss the process used to fit scaling laws to our empirical results. For each algorithm (Data-Parallel or DiLoCo with $M \in \{1, 2, 4, 8\}$) we ran extensive hyperparameter sweeps on models of size N up to 2.4B. To fit a scaling law for loss L as a function of N, we pick the best hyperparameters for each N, and fit a power law to $L(N) \approx AN^{\alpha}$. For DiLoCo, we fit two types of scaling laws: independent power law for each value of M and joint fits where we fit a single scaling law $L(N, M) \approx AN^{\alpha}M^{\beta}$. We also fit power laws, both individual and joint, to the optimal learning rate γ and batch size B. Due to a lack of space, we give the parameters of the power laws fit from our data in Section F. Here we overview some of the important findings derived from these scaling laws.

Interpolation. First, we measured whether individual or joint fit scaling laws *interpolated* to our data in DiLoCo better. We do this via leave-one-out validation. We fit scaling laws for L, γ, and B, but only using data up to $N = 1.3$B parameters, leaving out our data on N = 2.4B parameters. We then use the scaling law to predict the optimal value for L, γ, and B at N = 2.4B parameters, across different values of M. We measure the residual $\text{res}(y, \tilde{y}) = |\log(y) - \log(\tilde{y})|$ between our prediction $\tilde{y}$ and the actual y, and average them across M. The results are in Table 5. We see that the joint fit matches independent for L and B, but does better at predicting γ.

Table 5: Residuals for scaling law predictions at N = 2.4B and varying M. We compare independent and joint scaling laws in predicting loss L, inner learning rate γ, and global batch size B. For the average residuals, we highlight which of independent or joint achieved a lower residual.

	Fit	L	γ	B
$M = 1$	Independent	0.011	0.35	0.00088
	Joint	0.019	0.14	0.19
$M = 2$	Independent	0.0099	0.18	0.44
	Joint	0.013	0.29	0.28
$M = 4$	Independent	0.012	0.051	0.25
	Joint	0.0082	0.086	0.11
$M = 8$	Independent	0.014	0.62	0.076
	Joint	0.0076	0.23	0.19
Average over M	Independent	**0.012**	0.30	**0.19**
	Joint	**0.012**	**0.19**	**0.19**

Extrapolation. Next, we use the scaling laws for hyperparameters to predict optimal hyperparameters for N = 4B and 10B, for Data-Parallel and DiLoCo with $M \in \{1, 2, 4\}$. We train using those

hyperparameters, and measure how well our scaling laws extrapolated. We compare the evaluation loss of these models in Table 6. We also show how well these aligned with our actual scaling laws for loss in Figure 8. Our extrapolation reveals a few key points. First, the trends discussed in Section 4 continue to hold at larger model scales, where DiLoCo with $M = 1, 2$ does better than Data-Parallel. Second, as with our interpolation experiments, joint-fit power laws extrapolate better for DiLoCo.

Table 6: Evaluation results on 4B and 10B models, using hyperparameters predicted by individual and joint scaling laws. We highlight DiLoCo evaluation results that were better (ie. lower) than Data-Parallel. We see that while DiLoCo with $M = 2$ does better than Data-Parallel with either independent or joint fit rules, DiLoCo $M = 1$ only does better when using joint fit.

Algorithm	Fit Method	Loss	
		4B	10B
Data-Parallel	Independent	2.224	2.090
DiLoCo, $M = 1$	Independent	2.229	2.103
	Joint	**2.219**	**2.086**
DiLoCo, $M = 2$	Independent	**2.218**	**2.083**
	Joint	**2.220**	**2.086**
DiLoCo, $M = 4$	Independent	2.232	2.098
	Joint	2.230	2.096

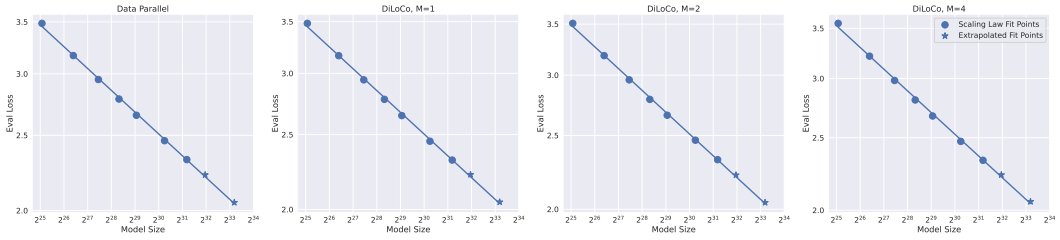


Figure 8: Scaling laws for Data-Parallel and DiLoCo. Pictured are both the loss values to form the scaling law (by training models up to a scale of 2.4B) and loss values attained on larger models (4B and 10B). While we present the individual-fit scaling laws for simplicity (and the ability to visualize Data-Parallel), the joint fit also predicts loss similarly well.

Other results. Due to a lack of space, we relegate other findings of our scaling laws to Section F. There we discuss the methodology in more detail, as well as the actual parameters of the power laws. We also discuss fitting other functional forms to the scaling laws, beyond simple power laws.

7 Discussion and Limitations

Our results above all show that like Data-Parallel, DiLoCo scales predictably with model size in ways that make it simpler to tune hyperparameters and train models at extremely large scales. Moreover, DiLoCo can offer significant benefits over Data-Parallel, including superior evaluation loss when using a single model replica, and increased optimal batch size for any number of model replicas. These benefits are robust to model scale, overtraining amount, and synchronization frequency.

While promising, there are clear limitations and directions for future exploration. First, while we have done our best to include downstream evaluations on our models (Section D), careful evaluation across domains and downstream tasks is important for validating scaling behavior. Moreover, there are a wide variety of datasets used for pre-training. While we have incorporated two widely used datasets of this kind in our analysis, we have not validated robustness across datasets. Second, like Hoffmann et al. [20], we use dense transformer architectures of varying sizes, and have not validated these results on alternative architectures (such as Mixture-of-Experts) or in other domains, notably multi-modal settings. Third, we use an idealized wall-clock model (Section B), leaving to future work a thorough analysis of runtime in practical systems, and how it varies according to the type of compute. Last,

while we have attempted to be comprehensive in analyzing the core DiLoCo algorithm, our work does not encompass various related methods, and does not constitute a comprehensive algorithmic benchmark. While we expect our findings transfer to related methods, explicit verification of this is left to future work.

References

[1] Kwangjun Ahn and Byron Xu. Dion: A communication-efficient optimizer for large models. *arXiv preprint arXiv:2504.05295*, 2025.

[2] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*, pages 26–33, 2001.

[3] Matt Beton, Matthew Reed, Seth Howes, Alex Cheema, and Mohamed Baioumy. Improving the efficiency of distributed training using sparse parameter averaging. In *ICLR 2025 Workshop on Modularity for Collaborative, Decentralized, and Continual Deep Learning*, 2025.

[4] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[5] Zachary Charles and Keith Rush. Iterated vector fields and conservatism, with applications to federated learning. In Sanjoy Dasgupta and Nika Haghtalab, editors, *Proceedings of The 33rd International Conference on Algorithmic Learning Theory*, volume 167 of *Proceedings of Machine Learning Research*, pages 130–147. PMLR, 29 Mar–01 Apr 2022. URL `https://proceedings.mlr.press/v167/charles22a.html`.

[6] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. *Advances in neural information processing systems*, 34:20461–20475, 2021.

[7] Zachary Charles, Nicole Mitchell, Krishna Pillutla, Michael Reneer, and Zachary Garrett. Towards federated foundation models: Scalable dataset pipelines for group-structured learning. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[9] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[10] Haotian Dong, Jingyan Jiang, Rongwei Lu, Jiajun Luo, Jiajun Song, Bowen Li, Ying Shen, and Zhi Wang. Beyond a single ai cluster: A survey of decentralized llm training. *arXiv preprint arXiv:2503.11023*, 2025.

[11] 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.

[12] Arthur Douillard, Qixuan Feng, Andrei A Rusu, Adhiguna Kuncoro, Yani Donchev, Rachita Chhaparia, Ionel Gog, Marc'Aurelio Ranzato, Jiajun Shen, and Arthur Szlam. DiPaCo: Distributed path composition. *arXiv preprint arXiv:2403.10616*, 2024.

[13] Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, et al. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

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

[15] FairScale authors. Fairscale: A general purpose modular pytorch library for high performance and large scale training. `https://github.com/facebookresearch/fairscale`, 2021.

[16] Louis Fournier, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky, and Edouard Oyallon. WASH: train your ensemble with communication-efficient weight shuffling, then average. *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2024. URL `https://arxiv.org/abs/2405.17517`.

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

[18] Qiaozhi He, Xiaomin Zhuang, and Zhihua Wu. Exploring scaling laws for local SGD in large language model training. *arXiv preprint arXiv:2409.13198*, 2024.

[19] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[21] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[22] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. GPipe: Easy scaling with micro-batch pipeline parallelism. *proceeding of Computer Science> Computer Vision and Pattern Recognition*, 2019.

[23] 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, 2024. URL `https://arxiv.org/abs/2412.01152`.

[24] Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. OpenDiLoCo: An open-source framework for globally distributed low-communication training. *arXiv preprint arXiv:2407.07852*, 2024.

[25] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

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

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

[28] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. PyTorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[29] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[30] Bo Liu, Rachita Chhaparia, Arthur Douillard, Satyen Kale, Andrei A Rusu, Jiajun Shen, Arthur Szlam, and Marc'Aurelio Ranzato. Asynchronous local-SGD training for language modeling. *arXiv preprint arXiv:2401.09135*, 2024.

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

[32] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[33] Alexandra Sasha Luccioni and Joseph D Viviano. What's in the box? a preliminary analysis of undesirable content in the common crawl corpus. *arXiv preprint arXiv:2105.02732*, 2021.

[34] Olvi L Mangasarian and Mikhail V Solodov. Backpropagation convergence via deterministic nonmonotone perturbed minimization. *Advances in Neural Information Processing Systems*, 6, 1993.

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

[36] 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.

[37] Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A Raffel. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36:50358–50376, 2023.

[38] Hiroki Naganuma, Xinzhi Zhang, Man-Chung Yue, Ioannis Mitliagkas, Philipp A Witte, Russell J Hewett, and Yin Tat Lee. Pseudo-asynchronous local sgd: Robust and efficient data-parallel training. *arXiv preprint arXiv:2504.18454*, 2025.

[39] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[40] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent, 2011. URL `https://arxiv.org/abs/1106.5730`.

[41] Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdEMAMix optimizer: Better, faster, older. *arXiv preprint arXiv:2409.03137*, 2024.

[42] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, February 2009. ISSN 0743-7315. doi: 10.1016/j.jpdc.2008.09.002. URL `https://doi.org/10.1016/j.jpdc.2008.09.002`.

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

[44] Alain Petrowski, Gerard Dreyfus, and Claude Girault. Performance analysis of a pipelined backpropagation parallel algorithm. *IEEE Transactions on Neural Networks*, 4(6):970–981, 1993.

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

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

[47] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=LkFG3lB13U5.

[48] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021.

[49] Keith Rush, Zachary Charles, Zachary Garrett, Sean Augenstein, and Nicole Elyse Mitchell. DrJAX: Scalable and differentiable mapreduce primitives in JAX. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024)*, 2024.

[50] Lorenzo Sani, Alex Iacob, Zeyu Cao, Royson Lee, Bill Marino, Yan Gao, Dongqi Cai, Zexi Li, Wanru Zhao, Xinchi Qiu, et al. Photon: Federated LLM pre-training. *arXiv preprint arXiv:2411.02908*, 2024.

[51] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. *arXiv preprint arXiv:2401.00448*, 2023.

[52] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-TensorFlow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

[53] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*, 2024.

[54] Sebastian U Stich. Local SGD converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[55] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, 2013. URL https://proceedings.mlr.press/v28/sutskever13.html.

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

[57] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021.

[58] 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. *International Conference on Machine Learning (ICML)*, 2023. URL https://openreview.net/forum?id=w2Vrl0zlzA.

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

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

[61] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[62] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32, 2019.

[63] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch FSDP: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

[64] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010.

A Related Work

Distributed training of LLMs. Due to their increasingly large sizes, the advancement of language models has necessitated advancements in distributed training methods. One vein of work uses data-parallel training, and attempts to shard its constituent computations across accelerators in efficient ways. This includes advancements in things like distributed data parallelism [52, 28], ZeRO parallelism [46, 48], fully-sharded data parallelism [15, 63], and pipeline parallelism [44, 22, 39]. Conceptual models of the impact of batch size on training time [35] aid in making the most effective trade-offs between training time and compute cost when using data-parallel training methods.

As scale continues to increase, the need to all-reduce gradients between data-parallel replicas becomes a bottleneck in training, causing accelerators to 'wait' on this allreduce for an unacceptably long time. This observation has motivated extensive work in pipeline parallelism and scheduling, communicating activations rather than gradients. An alternative line of work keeps the basic data processing pattern of data parallelism while directly minimizing communication requirements.

Three broad families of algorithms exist in that space: (1) sparse updates (including CocktailSGD [58], PowerSGD [56], DeMo [43], and Dion [1]), (2) fast asynchronous updates (including Hogwild [40], WASH [16], and Sparta [3]), and (3) infrequent updates (including LocalSGD [54], FedOpt [47], DiLoCo [11], and PALSGD [38]). These are extremely active areas of work, and we do not attempt to give a comprehensive survey of them all simultaneously. We instead defer the interested reader to a survey of decentralized LLM training [10].

In this work, we focus on the third category. Douillard et al. [11] showed that we can reduce communication costs in LLM training significantly by training multiple models independently with infrequent synchronization. Their method, DiLoCo, massively reduces communication overhead when training LLMs with a moderate numbers of model replicas. It has also shown great promise in training LLMs up to 10 billion parameters [24, 23]. This work has also been extended to asynchronous overlapped updates [30, 13], and low-communication expert sharding [12].

Federated learning. There is an enormous body of work on communication-efficient training methods for machine learning. In that vein, DiLoCo is closely related to algorithms used in federated learning to perform communication-efficient training over decentralized data, often (but not exclusively) on edge devices [25]. The prototypical algorithm used in federated learning, FedAvg [36], reduces communication costs by training models in parallel, with periodic model averaging. This algorithm has been invented and reinvented throughout machine learning, and is also known as Local SGD [54], parallel SGD [64], and parallel online backpropagation [34]. The use of inner and outer optimization steps (as in DiLoCo, see Algorithm 1) was first used by Hsu et al. [21] and Reddi et al. [47] for federated learning, focusing on SGD as the inner optimizer and SGDM or Adam [27] as the outer optimizer in order to leverage more sophisticated optimizers in resource-constrained settings. DiLoCo is also closely related to many other federated optimization methods, though the huge amount of work in this area makes it impossible to summarize succinctly here. We instead refer the interested reader to the survey of Wang et al. [57], though the field has of course progressed since then. While federated learning is often applied to more moderately sized models, Charles et al. [7] and Sani et al. [50] show that federated learning can be used to good effect for LLM training.

Scaling laws. Scaling laws work often aims to estimate how empirical generalization error scales with various facets, including model size and training set size. Empirical scaling analyses with power law behavior date have existed for decades (see [2]). Hestness et al. [19] developed power laws for model and dataset size across various tasks and model architectures (including encoder-decoder LSTM models). More recently, scaling laws for transformer-based LLMs were proposed by Kaplan et al. [26] and Hoffmann et al. [20], who exhibited power law relationships between LLM performance and model size. Sine then, there has been a large number of works developing scaling laws for other facets of LLMs, including (among many others) inference costs [51], data-constrained training [37], and overtraining [17].

Scaling laws for DiLoCo were previously studied by He et al. [18], who show that DiLoCo with 8 replicas exhibits analogous scaling behavior to Data-Parallel. He et al. [18] use a fixed number of replicas, batch size, and outer learning rate, and an unspecified total token budget.[4] Our work

[4] At the time of writing, He et al. [18] only say that "Each model was trained to achieve adequate convergence".

expands on many aspects of their work and explores others that were not considered, including but not limited to: $10\times$ larger models, varying the number of replicas (including single-replica DiLoCo), varying token budgets and overtraining, parametric function fitting, scaling laws for hyperparameters, and optimal batch size analysis.

B Wall-Clock Time Model

In this section, we present an idealized model for the wall-clock times of Data-Parallel and DiLoCo. We measure the total elapsed time, which means that parallelization (e.g. via increasing the batch size) reduces wall-clock time.

B.1 Computation Time

Here, we mean the time expended by floating point operations in model training, ignoring communication time across nodes (which we treat separately in the section below). We use the idealized model where the total FLOPs $C = 6ND$. Given some number of chips R, each of which can perform Q floating point operations per second, the total computation time is bounded below by C/RQ. The number of chips R is purely a function of model size N and global batch size B. The number of chips does not depend on the algorithm (Data-Parallel or DiLoCo) or number of model replicas when using DiLoCo.

B.2 Communication Time

The network connectivity is characterized by a bandwidth W and latency ϵ. When performing an all-reduce of N parameters over R compute nodes, the lower bound on the amount of traffic sent and received by at least one of the compute nodes participating in the all-reduce is $2N(1 - R^{-1})$ [42]. Such algorithms are called *bandwidth-optimal*. Since communication across the nodes is done synchronously but in parallel, in a network with bandwidth W and latency ε between each pair of nodes, the time to complete the all-reduce is at least

$$\frac{2N}{W}\left(1 - \frac{1}{R}\right) + \varepsilon.$$

DiLoCo [11] was designed for settings where models are too large to fit in a single datacenter, so they must be trained across compute islands connected by low bandwidth. To model this, we will assume that we are training over R compute nodes (typically, GPUs or TPUs). Some of these are connected by networks within a datacenter, and others are connected across datacenters. We let W_0, ε_0 denote the bandwidth and latency of the within-datacenter network, and W_1, ε_1 analogously defined for the cross-datacenter network. Typically, $W_0 \geq W_1, \varepsilon_0 \leq \varepsilon_1$.

Data-Parallel: At every training step T, we have to perform an all-reduce over all R compute nodes. Since some nodes are connected across datacenters, the total communication time is at least

$$\left(\frac{2N}{W_1}\left(1 - \frac{1}{R}\right) + \varepsilon_1\right) T$$

DiLoCo, $M = 1$: At every inner step T, we perform an all-reduce over all R devices as in Data-Parallel training. We also do an all-reduce every H steps for the outer optimization. Some of these nodes are connected across datacenters, so the communication time per all-reduce is at least $2N(1 - R^{-1})W_1^{-1} + \varepsilon_1$. The total communication time is therefore at least

$$\left(\frac{2N}{W_1}\left(1 - \frac{1}{R}\right) + \varepsilon_1\right) T \left(1 + \frac{1}{H}\right).$$

DiLoCo, $M \geq 2$: We assume that each of the M model replicas is trained on compute nodes connected within the same datacenter. In each inner step T, each model replica is trained by R/M devices which need to do an all-reduce. However, no communication occurs between datacenters, so the communication time of each inner step is bounded by $2N(1 - MR^{-1})W_0^{-1} + \varepsilon_0$.

Each outer optimization step involves all-reducing over all R devices, connected across datacenters. This incurs a communication time of at least $2N(1 - R^{-1})W_1^{-1} + \varepsilon_1$. Since it occurs only every H steps, the total communication time is bounded below by:

$$\left(\frac{2N}{W_0}\left(1 - \frac{M}{R}\right) + \varepsilon_0\right)T + \left(\frac{2N}{W_1}\left(1 - \frac{1}{R}\right) + \varepsilon_1\right)\frac{T}{H}.$$

Note that this suggests that as long as $H \geq W_0/W_1$, the outer communication steps incur at most half of the total communication cost.

Streaming DiLoCo. We note that the computed cost above applies to the Streaming DiLoCo [13] as well. While the inner step remains the same, the outer step is smoothed such that each fragment $p \in \{1, \ldots, P\}$ is every H steps. However, fragment communication is offset such that some fragment is communicated every H/P steps, resulting in the communication amortizing to the same per-step cost. This is expected as Streaming DiLoCo reduces peak communication over any step, but does not reduce total communication across training.

Overlapping communications. Another contribution of Douillard et al. [13] is the ability to overlap communications required for the outer optimizer with computation by using a stale version of the fragment in the outer optimizer, and merging the result of this outer optimization with the locally optimized fragment. This would allow, for example, the communication term to be omitted from the calculation for wall-clock-time, if computation time dominates communication time. This setting is different from an algorithmic perspective, so its impact on scaling would need to be examined independently.

B.3 Total Wall-Clock Time

The total wall-clock time is a sum of the computation and communication times above. To measure the communication time, we must know the number of chips R used for each experiment, the number of FLOPs per chip per second Q, the bandwidth and the latency of the within-datacenter and cross-datacenter networks. For computation costs, we use a slightly idealized number of chips R based on our experiments, but ensuring that doubling the global batch size would double the number of chips. We base the constant Q on publicly available information about the FLOPs capabilities of the TPU v5e and v6e chips[5], which have peak compute per chip (in bfloat16) of 197 teraflops and 918 teraflops, respectively. Assuming a maximum FLOPs usage of 50%, these chips have an actual compute of approximately 100 and 408 teraflops, respectively. When computing idealized compute time, we set $Q = 300$ teraflops, somewhere in-between the two.

For bandwidth and latency, we consider three archetypes of networks:

1. **High-bandwidth network** with bandwidth $W_{\text{high}} = 400$ gigabits per second and a latency of $\varepsilon_{\text{high}} = 10^{-4}$ seconds.
2. **Medium-bandwidth network** with bandwidth $W_{\text{med}} = 100$ gigabits per second and a latency of $\varepsilon_{\text{med}} = 10^{-3}$ seconds.
3. **Low-bandwidth network** with bandwidth $W_{\text{low}} = 10$ gigabits per second and a latency of $\varepsilon_{\text{low}} = 10^{-2}$ seconds.

We stress that these are not based on any actual systems, and are simply designed as instructive archetypes of networks. For the idealized communication time, we always use the high-bandwidth network for the within-datacenter network, and one of the three for the cross-datacenter network.

C Datasets and Licenses

In this section, we discuss the datasets used in our experiments and their respective licenses. We note that the datasets themselves are all referenced in Section 3.

[5] See https://cloud.google.com/tpu/docs/v6e.

- C4 [45]. This dataset is a cleaned version of Common Crawl's web crawl corpus[6]. We used the version accessible via Hugging Face, provided by Allen AI[7], and made available under the ODC-BY license. Because it is derived from Common Crawl, its usage is also bound by the Common Crawl terms of use. We use the **en** version of the dataset. We note that while widely used and filtered, the dataset still contains undesirable content including hate speech. See [33] for an analysis of its contents, and discussion of mitigation strategies. We do not release any assets derived from this dataset.
- Dolma [53]. This dataset consists of 3 trillion tokens drawn from web content, academic publications, code, books, and related material. We used the **v1_7** version of the dataset accessible via Hugging Face, provided by Allen AI[8], and made available under the ODC-BY license. We note that the usage of this dataset is also bound by license agreements and restrictions of its original data sources. See [53] for more details.
- HellaSwag [61]. We use this dataset for zero-shot evaluation metrics. We used the version available under the MIT license[9] For details, see `https://rowanzellers.com/hellaswag/`.
- Piqa [4]. We use this dataset for zero-shot evaluation metrics. We used the version available under the Academic Free License v3.0[10].
- Arc-Easy [9]. We use the Arc-Easy dataset for zero-shot evaluation metrics. We use the version accessible via Hugging Face, provided by Allen AI[11], made available under the CC-BY-SA-4.0 license.

D Additional Experimental Results

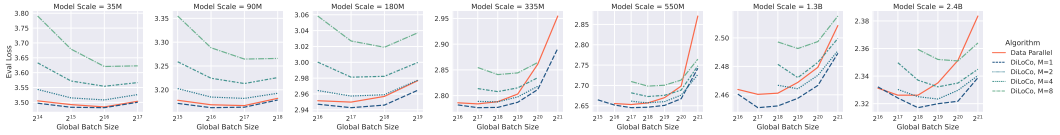


Figure 9: Evaluation loss of Data-Parallel and DiLoCo as a function of global batch size (in tokens).

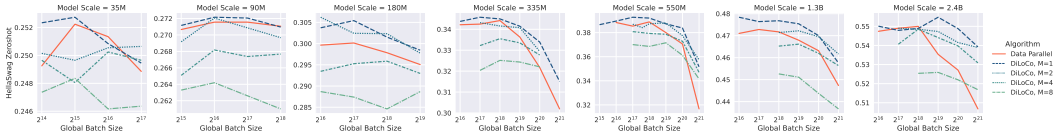


Figure 10: Zero-shot evaluation accuracy on HellaSwag of Data-Parallel and DiLoCo as a function of global batch size (in tokens).

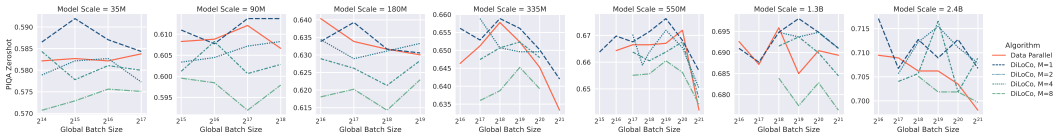


Figure 11: Zero-shot evaluation accuracy on Piqa of Data-Parallel and DiLoCo as a function of global batch size (in tokens).

In this section, we give additional experimental results that expand on those in Section 4. In Figures 9, 10, 11, and 12, we present evaluation loss and evaluation accuracy on various downstream zero-shot

[6] `https://https://commoncrawl.org/`
[7] `https://huggingface.co/datasets/allenai/c4`
[8] `https://huggingface.co/datasets/allenai/dolma`
[9] `https://github.com/rowanz/hellaswag`.
[10] `https://github.com/ybisk/ybisk.github.io/tree/master/piqa`
[11] `https://huggingface.co/datasets/allenai/ai2_arc`

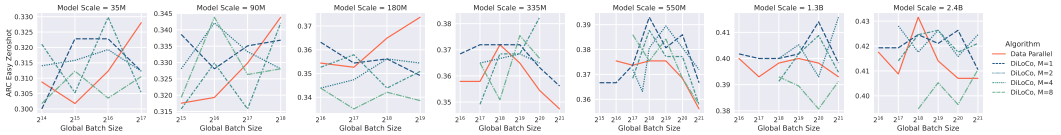
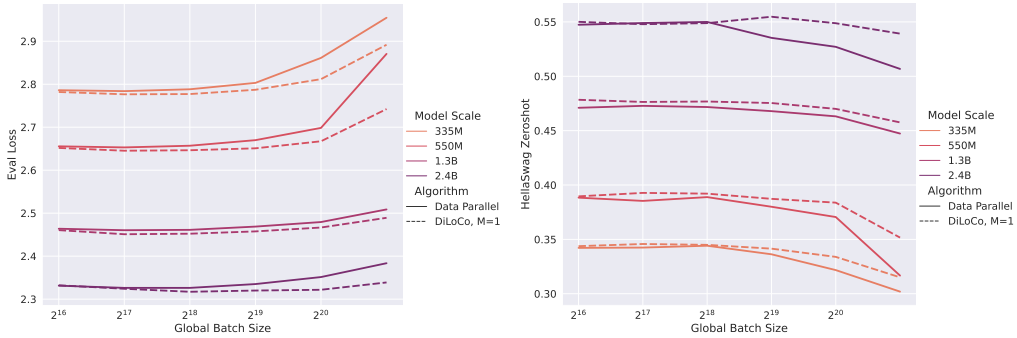


Figure 12: Zero-shot evaluation accuracy on Arc-Easy of Data-Parallel and DiLoCo as a function of global batch size (in tokens).

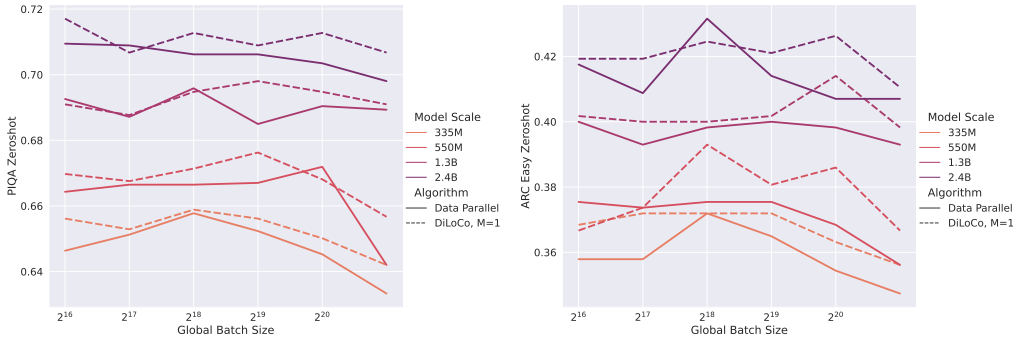
tasks, as a function of algorithm, model size, and global batch size. The results consistently show that Data-Parallel's evaluation performance degrades quickly as batch size increases. By contrast DiLoCo's performance degrades more slowly, or even improves, as batch size increases. We note that Arc-Easy was quite noisy as an evaluation task.



(a) Evaluation loss.

(b) Zero-shot accuracy on HellaSwag.

Figure 13: Evaluation loss and zero-shot accuracy of Data-Parallel and DiLoCo with $M = 1$ for varying model and global batch sizes (measured in tokens). In all settings, DiLoCo with $M = 1$ does better than Data-Parallel, and the gap between them increases with batch size.



(a) Zero-shot accuracy on Arc-Easy.

(b) Zero-shot accuracy on Arc-Easy.

Figure 14: Zero-shot evaluation accuracy of Data-Parallel and DiLoCo with $M = 1$ for varying model and global batch sizes (measured in tokens), on Piqa and Arc-Easy. In nearly all settings, DiLoCo with $M = 1$ does better than Data-Parallel, and the often the gap increases with batch size.

In Figures 13 and 14, we compare Data-Parallel and DiLoCo with $M = 1$ in terms of their evaluation loss and zero-shot evaluation accuracy on HellaSwag, Piqa and Arc-Easy. As above, we note that DiLoCo with $M = 1$ has an improved tolerance to larger batch sizes.

E Ablations

Here, we expand upon the ablation studies discussed in Section 5. We give more detailed experimental analyses, as well as idealized wall-clock times under varying networks.

E.1 Synchronization Cadence

As discussed in the synchronization cadence ablations in Section 5, we perform an ablation over H for varying M and model sizes N. For each setting, we take the optimal inner learning rate and global batch size from above, and sweep H over $\{1, 5, 10, 30, 100, 300\}$ and η over $\{0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

In addition to the results in Section 5, we analyze how H impacts the evaluation loss of DiLoCo in Figure 15. For all models, synchronizing every step ($H = 1$) performs the worst, but after this point all values of H perform somewhat comparably. For a fixed N and M, evaluation loss increases as H increases. However, this increase is less pronounced for $M = 1$ and larger models. This yields an important finding: As the model size N increases, we can actually perform synchronization across DiLoCo replicas less frequently, while nearly maintaining evaluation performance.

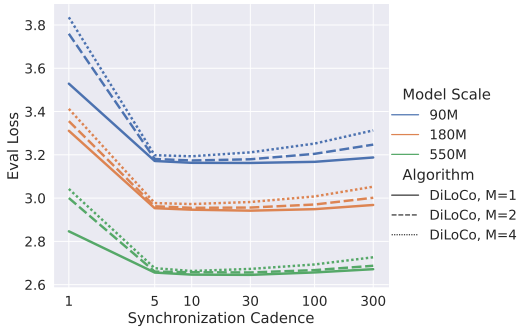
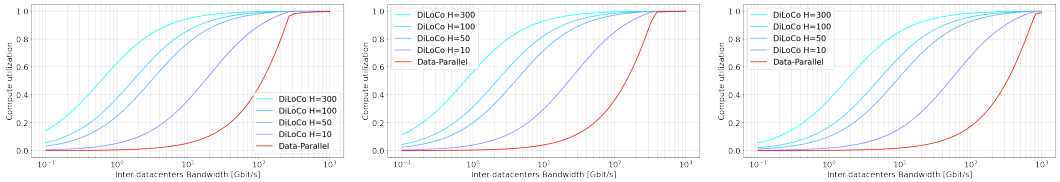


Figure 15: **Infrequent synchronization works better for larger models.** Outside of $H = 1$, which performs the worst, evaluation loss increases with H. However, the rate of increase is less pronounced for DiLoCo with $M = 1$ and for larger models, suggesting that large models can be synchronized quite infrequently.

Table 7: **Simulated compute utilization**. We estimate the step time based on the required flops using the rule proposed by Kaplan et al. [26] and a max flop utilization of 60%. We estimate the bandwidth (in Gbit/s) required to reach a level of compute utilization using [13]'s simulator. We highlight in light blue 10× reduction of bandwidth, and in dark blue 100× reduction.

Architecture	Size	Step time	Method	Gbit/s to reach a compute utilization CU =? 50%	80%	90%	95%	99%
Chinchilla	10B	0.8s	Data-Parallel	104.8	184.2	222.3	222.3	390.7
			DiLoCo, $H = 1$	104.8	184.2	222.3	222.3	390.7
			DiLoCo, $H = 10$	16.0	49.4	86.8	152.6	222.3
			DiLoCo, $H = 50$	**3.0**	**11.0**	23.3	41.0	126.5
			DiLoCo, $H = 100$	**1.4**	**6.2**	**13.3**	23.3	86.8
			DiLoCo, $H = 300$	**0.5**	**2.0**	**4.3**	**9.1**	41.0
Llama3	405B	26s	Data-Parallel	126.5	222.3	268.3	323.8	323.8
			DiLoCo, $H = 1$	126.5	222.3	268.3	323.8	323.8
			DiLoCo, $H = 10$	19.3	72.0	126.5	184.2	268.3
			DiLoCo, $H = 50$	**3.6**	**13.3**	28.1	59.6	184.2
			DiLoCo, $H = 100$	**2.0**	**7.5**	**16.0**	33.9	126.5
			DiLoCo, $H = 300$	**0.7**	**3.0**	**6.2**	**13.3**	59.6
DeepSeek-V3	671B	20s	Data-Parallel	323.8	569.0	686.6	686.6	1000.0+
			DiLoCo, $H = 1$	323.8	569.0	686.6	686.6	1000.0+
			DiLoCo, $H = 10$	49.4	152.6	268.3	390.7	686.6
			DiLoCo, $H = 50$	**7.5**	**33.9**	72.0	126.5	390.7
			DiLoCo, $H = 100$	**4.3**	**16.0**	**41.0**	72.0	268.3
			DiLoCo, $H = 300$	**1.7**	**6.2**	**13.3**	**28.1**	126.5

Compute utilization. The synchronization cadence of DiLoCo is critical to training large-scale models distributed across the world. Indeed, less frequent synchronization (larger H) diminishes



(a) 10B Chinchilla (b) 405B Llama3 (c) 671B DeepSeek-V3

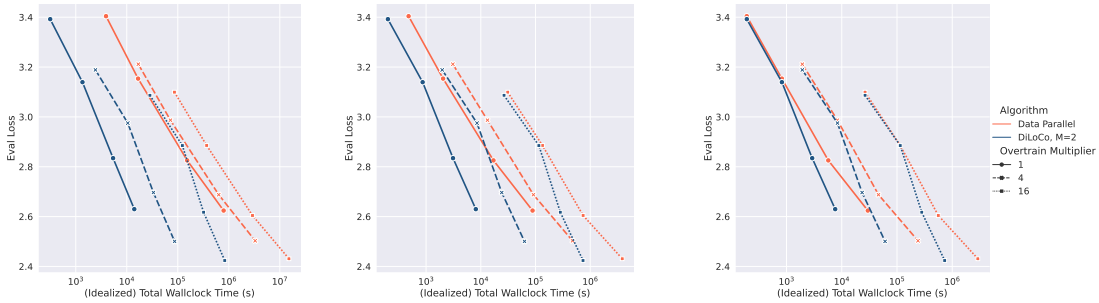
Figure 16: **DiLoCo greatly increases compute utilization**. Here we present simulated compute utilization for DiLoCo and Data-Parallel across a range of bandwidth and synchronization cadences H. A compute utilization of 0.8 means 80% of the time is spent in computation, and 20% in communication. A higher value is better. We see similar results for other model sizes, but omit for visual clarity.

the bandwidth requirements of training. Following [13], we simulate the amount of bandwidth required to have a compute utilization ($\frac{\text{compute time}}{\text{total time}}$) as large as possible for three types of LLMs: a 10B Chinchilla-style transformer [20] in Fig.16a, a 405B Llama3 model [14] in Fig.16b, and a 671B DeepSeek-v3 MoE [29] in Fig.16c. We also report raw numbers in Table 7.

E.2 Overtraining

We expand on the overtraining ablation study in Section 5. As we showed there, DiLoCo continues to enjoy large optimal batch sizes, compared to Data-Parallel. This improves horizontal scalability, decreasing end-to-end wallclock time. Wallclock time is also smaller for $M > 1$ due to reduced communication.

To illustrate this, we plot idealized training time of Data-Parallel and DiLoCo with $M = 2$ for different overtraining amounts in Figure 17. DiLoCo speeds up overtraining by reducing communication costs, and utilizing larger batch sizes, therefore requiring fewer serial training steps. This suggests that DiLoCo is a significant boon in overtraining, as we can amortize compute time (which can be quite long for overtraining) via horizontal scalability.



(a) Network with a bandwidth of 10 gigabits/s and a latency of 10^{-2} seconds (**low-bandwidth**). (b) Network with a bandwidth of 100 gigabits/s and a latency of 10^{-3} seconds (**medium-bandiwdth**). (c) Network with a bandwidth of 400 gigabits/s and a latency of 10^{-4} seconds (**high-bandwidth**).

Figure 17: For Data-Parallel and DiLoCo, $M = 2$, we plot idealized wall-clock time (see Section B) for training by Data-Parallel and DiLoCo, $M = 2$ across compute nodes connected via high-, medium-, and low-bandwidth networks. For each algorithm and overtraining amount, we display lines represent varying model sizes, from 335M parameters to 2.4B. DiLoCo is faster in all settings, due to both its reduced communication and its tolerance to larger batch sizes. Even in the high-bandwidth setting, the larger batch sizes increase horizontal scalability, reducing end-to-end wallclock time. We see similar results for $M \geq 4$ and for smaller models, but omit for visual clarity.

F Scaling Laws

We now discuss the process we used to fit scaling laws to our empirical results, expanding on the discussion in Section 6. Recall that for each algorithm (Data-Parallel or DiLoCo with $M \in \{1, 2, 4, 8\}$) we ran extensive hyperparameter sweeps on models of size N up to 2.4B. To fit a scaling law for loss L as a function of N, we pick the best hyperparameters (in terms of evaluation loss) for each N, aggregate the loss values, and fit some kind of function for $L(N)$, such as a power law [26]. We will also fit scaling laws to the optimal hyperparameters. We will fit two types of scaling laws for DiLoCo: independent fits for each value of M, and joint fits where we fit a single scaling law as a function of N and M simultaneously.

F.1 Independent scaling laws

Scaling laws for loss. We first fit scaling laws for the loss obtained by Data-Parallel training. We fit a power law to the evaluation loss of Data-Parallel as a function of N via the power law approximation $L(N) \approx AN^{\alpha}$. Note that this can easily be done via applying linear fit techniques to $\log(L)$, and is not sensitive to things like initial values of A, α. The resulting power law is in the first row of Table 8.

We mirror this above for DiLoCo when doing independent fits. For each value of N, M, we record the lowest loss value across all hyperparameters. We can then fit power law $L_M(N) := L(N, M) \approx AN^{\alpha}$ for each M. The results are given in Table 8.

Table 8: Power law approximations for loss $L(N) \approx AN^{\alpha}$.

	A	α
Data-Parallel	18.129	-0.0953
DiLoCo, $M = 1$	18.363	-0.0961
DiLoCo, $M = 2$	18.768	-0.0969
DiLoCo, $M = 4$	19.762	-0.0992
DiLoCo, $M = 8$	21.051	-0.1018

The results show that Data-Parallel and DiLoCo see similar predicted reductions in loss as a function of N. Notably, the fit parameters suggest that DiLoCo, $M = 1$ outperforms Data-Parallel at essentially all but the absolute smallest model scales. This mirrors the results discussed in Section 4.

Scaling laws for hyperparameters. For Data-Parallel, we fit scaling laws for learning rate γ and batch size B. For DiLoCo, we fit scaling laws for inner learning rate γ and global batch size B. Given their analogous role in the algorithms, we fit them in the same way. For (inner) learning rate, we use the same approach as fitting scaling laws for loss: for each N (and M, for DiLoCo), we select the best hyperparameters, and fit a power law. The results are in Table 9.

For (global) batch size, we alter this slightly. As discussed in Section 3, our sweeps use powers of 2 for batch size, in order to saturate compute. However, the optimal batch size may be between these values. To account for this, we first fit a quadratic approximation to the batch size. Specifically, for each value of N we look at the loss as a function of $\log_2(B)$ (when using the best learning rate for that B), and fit a quadratic to this function. We select the minima of those quadratics and fit a power law to them, as a function of N. The results are in Table 10.

DiLoCo has a third hyperparameter we could fit scaling laws to: the outer learning rate. However, as shown in Section 4, the optimal outer learning rate is (for sufficiently large models) seemingly constant. Therefore, a scaling law would seemingly not yield any improved predictive performance over simply using the best outer learning rate for each M (see Figure 5).

F.2 Joint scaling laws

Alternatively, we can fit joint power laws to various facets of DiLoCo, using a two-variable power law $f(N, M) \approx AN^{\alpha}M^{\beta}$. We do this for loss L, inner learning rate γ and global batch size B. For the first two, we select, for each value of N, M, the best learning rate and loss. For batch size we do the same, but using the quadratic approximations from the section above. We can then fit a joint

Table 9: Power law approximations for (inner) learning rate $\gamma(N) \approx AN^{\alpha}$.

	A	α
Data-Parallel	16319.2	-0.819
DiLoCo, $M = 1$	74620.6	-0.945
DiLoCo, $M = 2$	3978.82	-0.780
DiLoCo, $M = 4$	4512.99	-0.789
DiLoCo, $M = 8$	618986	-1.102

Table 10: Power law approximations for (global) batch size $B(N) \approx AN^{\alpha}$.

	A	α
Data-Parallel	462.68	0.281
DiLoCo, $M = 1$	27.873	0.435
DiLoCo, $M = 2$	15.749	0.479
DiLoCo, $M = 4$,	10.957	0.510
DiLoCo, $M = 8$	38.072	0.455

Table 11: Joint power law approximations $f(N, M) = AN^{\alpha}M^{\beta}$ for the loss L, inner learning rate γ, and batch size B of DiLoCo.

	A	α	β
L	19.226	-0.0985	0.0116
γ	22256	-0.8827	0.2929
B	14.521	0.4695	0.3399

power law via standard linear regression techniques. The resulting power laws are in Table 11. Just as with the independent fits, we do not attempt to fit scaling laws to the outer learning rate η, as the optimal value is independent of N.

F.3 Measuring goodness-of-fit

Now that we have two different ways of developing scaling laws for DiLoCo, we can attempt to ask which one yields better predictions. First we do this via leave-one-out validation. Specifically, we use the same methodology as above to fit scaling laws for L, γ, and B, but only using data up to $N =$ 1.3B parameters, leaving out our data on N = 2.4B parameters. We then use the scaling law to predict the optimal value for L, γ, and B at N = 2.4B parameters, across different values of M.

Table 12: **Joint fit scaling laws match or beat independent fit.** Here we give the residuals for scaling law predictions for N = 2.4B and varying M. We compare the residual of independent and joint fitting strategies in predicting loss L, inner learning rate γ, and global batch size B. For the average residuals, we highlight which of independent or joint achieved a lower residual. We see that the joint fit matches independent for L and B, but does better at predicting γ.

	Fit	L	γ	B
$M = 1$	Independent	0.011	0.35	0.00088
	Joint	0.019	0.14	0.19
$M = 2$	Independent	0.0099	0.18	0.44
	Joint	0.013	0.29	0.28
$M = 4$	Independent	0.012	0.051	0.25
	Joint	0.0082	0.086	0.11
$M = 8$	Independent	0.014	0.62	0.076
	Joint	0.0076	0.23	0.19
Average over M	Independent	**0.012**	0.30	**0.19**
	Joint	**0.012**	**0.19**	**0.19**

Given a predicted value $\tilde{y}$ and a reference value of y, we compute the *residual* of our prediction as the mean absolute error of the logarithm: $\text{res}(y, \tilde{y}) = |\log(y) - \log(\tilde{y})|$. We use this measure as it works well for all three variables simultaneously, despite the fact that they vary greatly in their scale. For each $M \in \{1, 2, 4, 8\}$, we compute the predicted value of the three parameters above, and measure

the residual relative the actual optimal value at N = 2.4B. We report these values, as well as their average across M, in Table 12.

Our results generally show that both approach is generally valid (as there is no clear winner) but also that there is significant variation in residuals between M. That being said, we see that on average, while the individual fit is slightly better at predicting the loss and global batch size, the independent fit is significantly better at predicting inner learning rate.

F.4 Extrapolating to larger models

We use the independent and joint fits to predict optimal hyperparameters for Data-Parallel and DiLoCo with $M \in \{1, 2, 4\}$ at 4B and 10B model scales. Note that for Data-Parallel, we can only use independent fits. We run training on these models with these hyperparameters, using a Chinchilla-optimal token budget, and compare the results.

Table 13: **Joint fit hyperparameters extrapolate well to larger models.** Here we show the evaluation results on 4B and 10B models, using hyperparameters predicted by individual and joint scaling laws. We highlight DiLoCo evaluation results that were better (ie. lower) than Data-Parallel. We see that while DiLoCo with $M = 2$ does better than Data-Parallel with either independent or joint fit rules, DiLoCo $M = 1$ only does better when using joint fit.

Algorithm	Fit Method	Loss	
		4B	10B
Data-Parallel	Independent	2.224	2.090
DiLoCo, $M = 1$	Independent	2.229	2.103
	Joint	**2.219**	**2.086**
DiLoCo, $M = 2$	Independent	**2.218**	**2.083**
	Joint	**2.220**	**2.086**
DiLoCo, $M = 4$	Independent	2.232	2.098
	Joint	2.230	2.096

We see two important facets. First, unlike results above, at 4B and 10B scales we see that DiLoCo with $M = 2$ actually outperforms both Data-Parallel and DiLoCo, $M = 1$, regardless of using individual or joint fit approaches. Second, we see that DiLoCo, $M = 1$ requires the joint fit to do better than Data-Parallel. Other than in this case, joint and independent fits perform comparably throughout. All in all, the joint fit approach to hyperparameters appears to have a slight edge over individual fit in extrapolating. Combined with its ability to also extrapolate to larger M, we generally recommend the joint fit approach for all hyperparameters.

We now use these loss values to see how they compare to the scaling laws fit above. We generally find that the loss values are predicted very well, within a few percentage points of the loss predicted by the scaling laws. We present the fit scaling law and extrapolated loss values in Figure 18.

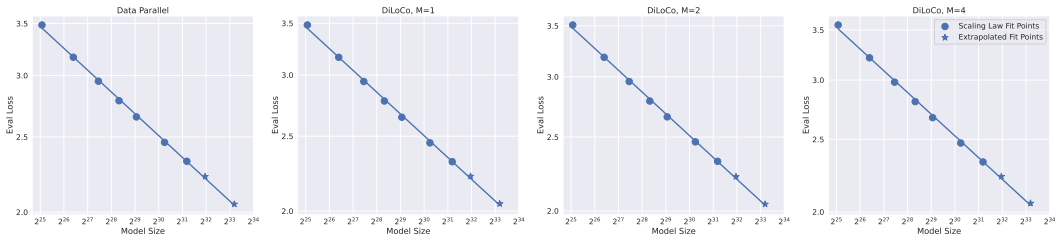


Figure 18: **DiLoCo scaling laws extrapolate well to larger models.** We present loss scaling laws for Data-Parallel and DiLoCo. Pictured are both the loss values to form the scaling law (by training models up to a scale of 2.4B) and loss values attained on larger models (4B and 10B). While we present the individual-fit scaling laws for simplicity, the joint fit also predicts loss well.

F.5 Parametric Function Fitting

Various works on scaling laws have often found it useful to fit functions more complex than power laws to the data [20]. We are particularly interested in parametric forms for our joint scaling laws. While Hoffmann et al. [20] use a risk decomposition argument to decompose loss as a function of N and D, it is not immediately clear how to do decompose $L(N, M)$. To that end, we use an empirical approach. We develop candidate functions, and determine which does best in an extrapolative sense. We use the following functional forms:

1. $L(N, M) \approx AN^{\alpha}M^{\beta}$
2. $L(N, M) \approx AN^{\alpha}M^{\beta} + C$
3. $L(N, M) \approx AN^{\alpha+\beta M} + C$
4. $L(N, M) \approx AN^{\alpha} + BM^{\beta} + C$

The first is included for comparison's sake, as it recovers the power law scaling law used above. Fitting more sophisticated functions can be much more sensitive to initial values of parameters, and also sensitive to outlier data. Therefore, when fitting these functions to our loss values above, we use the general strategy proposed by Hoffmann et al. [20].

In detail, let $f_Q(N, M)$ denote one of the functional forms above, where Q represents the set of parameters to be fit (e.g. $Q = \{A, \alpha, \beta\}$ for the first). Let Huber_δ denote the Huber loss with parameter δ. Let $\mathcal{N}, \mathcal{M}$ denote the set of values of N and M considered. For each N, M, we have an empirical loss $L(N, M)$, and some estimate of the loss $f_Q(N, M)$. We then solve the following minimization problem:

$$\min_Q \sum_{N\in\mathcal{N}} \sum_{M\in\mathcal{M}} \text{Huber}_\delta\Big(\log f_Q(N, M) - \log L(N, M)\Big).$$

We minimize this via L-BFGS, using some initialization Q_0 for the parameters. We repeat this process for 256 random initializations Q_0 of the parameters. We hold out all loss values at the N = 2.4B scale, and select the parameters Q that best fit the held-out data, measured in terms of the average residual $|\log f_Q(N, M) - \log L(N, m)|$ over all M.

Table 14: **Parametric function fitting improves joint scaling laws.** We showcase various parametric approximations to the empirical loss function $L(N, M)$, along with their validation error on held-out loss data at the N = 2.4B scale. We see that joint power laws (the first) and classical loss decomposition (the last) are significantly worse at predicting loss on the held-out data.

Parametric form	Average Residual
$L(N, M) \approx AN^{\alpha}M^{\beta}$	0.0044
$L(N, M) \approx AN^{\alpha}M^{\beta} + C$	0.0035
$L(N, M) \approx AN^{\alpha+\beta M} + C$	**0.0025**
$L(N, M) \approx AN^{\alpha} + BM^{\beta} + C$	0.0043

We see that the power law (row 1) and additive decomposition (row 4) are significantly worse at extrapolating loss values than more nuanced parametric forms. We note that the additive decomposition resembles the decomposition of loss as a function of model size N and token budget D used by Hoffmann et al. [20], but does not seem to reflect how M affects loss for DiLoCo. We leave it as an open problem to determine what parametric forms better predict loss, and can be explained by theoretical understanding of communication-efficient training.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We give extensive empirical evidence explicitly backing each claim made in the abstract and introduction in Sections 4, 5, and 6, with further evidence in Appendix D, E, and F.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We explicitly identify limitations of our work in Section 7. We have attempted to be absolutely transparent about every detail in our empirical work, as a way to make clearer the limitations, as well.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [NA]

Justification: We do not include any theoretical results in the work.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: We explicitly discuss every aspect of our empirical setup in Section 3. Our scaling law methodology is also explicitly detailed in Section F, and our wallclock time model is detailed exactly in Section B.

Guidelines:

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

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

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

   Answer: [No]

   Justification: The datasets used for our experiments are available publicly, through standard distribution channels (including Hugging Face). We do not explicitly open-source our code. While we would like to make the code available (and are working on a modified version of it that can be open-sourced), we could not open-source the code as is due to the use of institution-specific code for launching experiments across accelerators. However, our code is based entirely on open-source libraries that we actively reference, including NanoDO [31] and DrJAX [49]. There is no special behavior in our experiments that is not detailed explicitly in Section 3. Indeed, our results strongly mirror those in other work, including [24].

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We specify this exactly in Section 3. We have detailed all possible aspects of training, including datasets, splits, optimizers, hyperparameters, model architecture details, model sizes, vocabulary size, and sequence lengths. Any missing details are unintentional, and would gladly be provided by the authors.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

Justification: While we would gladly produce and report error bars, doing so to within any statistically significant degree would be prohibitively expensive due to the large number of hyperparameters and model sizes used throughout.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We detail the exact accelerators used for our experiments. We also detail what aspects of their computational profile were used for our idealized wall-clock computations in Section B.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have evaluated our work against each of the concerns listed in the guidelines, and believe that it adheres to the provided standard.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Generally, our work has no specific possible societal impact of the sort in the guidelines below. Algorithms like DiLoCo are similar to the example of "optimizing neural networks faster" in the guidelines below. That being said, we do wish to give one specific societal impact that is relevant: communication-efficient algorithms like DiLoCo enable thte training of models in a decentralized fashion. This means that some assumptions about AI used in things like policy-making may not hold, such as the assumption that LLMs are trained within a single datacenter. For example, AI policy monitoring based on total amount of compute may be more difficult in settings where algorithms like DiLoCo are deployed, as the amount of compute used by each compute node may be quite small, and the nodes themselves may be decentralized. It may also allow smaller groups and institutions to do LLM training at scale. This can be good (democratizing access to LLMs, and enabling greater competition in AI marketplaces) and bad (enabling adversarial actors to scale up training of LLMs), though the specifics will depend on how the algorithms evolve, and by what margin they can reduce communication and computation bottlenecks.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [NA]

    Justification: We release no data or models as part of this work, and do not believe that the work poses any risks in this regard.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring

that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We discuss all datasets used in Section 3. We further detail the datasets, versions, and licenses used in Section C.

    Guidelines:

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

13. **New assets**

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

    Answer: [NA]

    Justification: We do not produce, disseminate, or release any new assets as part of this work.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: Our work did not involve any crowdsourcing or research with human subjects.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: Our work did not involve any crowdsourcing or research with human subjects, and did not necessitate any IRB approvals.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [NA]

    Justification: We do not explicitly discuss this in the main body of the work, but we did not use LLMs for any component of the research itself. The methodology, coding, and initial draft were all completed entirely by humans. LLMs were only used for minor stylistic and grammatical edits after the first draft.

    Guidelines:

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