Can Model Merging Improve Aggregation in DiLoCo?

Stefan Horoi[1,2*] **Benjamin Thérien**[1,2] **Guy Wolf**[1,2] **Eugene Belilovsky**[2,3]
[1]Université de Montréal [2]Mila – Quebec AI Institute [3]Concordia University

Abstract

Model merging techniques, which aggregate independently finetuned models into one to combine their capabilities, have become a topic of significant interest in recent years, with a broad array of methods having been proposed to tackle this problem. Simultaneously, an emerging trend in distributed learning has been the use of methods such as local SGD and DiLoCo, which greatly reduce communication costs by periodically aggregating the independently trained local models. However, these communication-efficient methods have been shown to degrade in performance relative to the FLOP-matched data-parallel gold standard as the number of independent local models grows and as the number of local training steps before global communication is increased. In this work, we draw an explicit analogy between the pseudo-gradient aggregation step in local SGD/DiLoCo and task arithmetic-based model merging, establishing a straightforward way to utilize merging methods in the context of distributed optimization. We then evaluate multiple state-of-the-art model merging methods in this setting and identify one method in particular, Iso-C, as a promising approach for improving DiLoCo. We find that DiLoCo SGD with Iso-C aggregation outperforms not only simple pseudo-gradient averaging but even the momentum-based DiLoCo, despite lacking a momentum mechanism itself. Building on this finding, we propose IsoLoCo, which adapts Iso-C for distributed training by equipping it with Nesterov momentum. Our empirical evaluations on language model pre-training across varying numbers of local workers show that IsoLoCo significantly outperforms DiLoCo, with the gap between them widening as the number of workers increases. This advantage remains present across model sizes and inner step counts, confirming that merging-inspired aggregation is an effective strategy for low-communication distributed training.

1 Introduction

The recent success of large language models (LLMs) has relied in large part on the scaling of model parameters, dataset size and amount of compute used for training. However, training models with parameter counts in the billions or trillions on internet-scale datasets comes with significant engineering challenges. The standard practice is to use data-parallel training, where multiple replicas of the model process mini-batches of data in parallel and their gradients are aggregated to update model parameters via gradient-based optimization. This requires large clusters of tightly co-located accelerators connected with high-bandwidth links, along with careful engineering to orchestrate communication and ensure efficient resource utilization. Furthermore, such synchronized training is vulnerable to device failures and makes poor use of heterogeneous hardware with varying processing speeds or network topologies.

Recently, [6] introduced Distributed Low-Communication (DiLoCo) training, a variant of FedAvg [24] and FedOpt [30] with AdamW [21] as the inner optimizer and stochastic gradient

*Correspondence to: stefan.horoi@umontreal.ca, eugene.belilovsky@concordia.ca
Code available at: https://github.com/shoroi/isoloco

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

descent (SGD) with Nesterov momentum [26, 32] as the outer optimizer. DiLoCo enables the training of deep learning models on multiple "islands" of poorly connected computational clusters.

Each cluster, or *worker*, trains its own model replica through multiple local optimization steps before sharing the accumulated parameter changes, i.e. *pseudo-gradients*, with the remaining clusters for a global optimization step. Performing many *inner* optimization steps between each *outer* synchronization allows DiLoCo to reach comparable final performance to data-parallel training while communicating significantly less. While highly effective at reducing communication costs and even surpassing data-parallel training at small worker counts, DiLoCo's final performance degrades in FLOP-matched settings as the number of workers or inner steps increases, making it difficult to scale without losing FLOP-efficiency [4, 33, 6, 31].

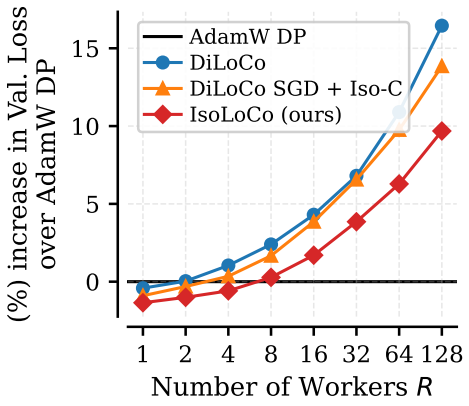

Figure 1: **IsoLoCo scales better with workers than DiLoCo.** Final validation loss is shown versus worker count for a 178M LLaMa-2 model in a FLOP-matched setting.

In parallel, as strong foundation models have been built and their weights made accessible to the public, these models have been adapted to many downstream tasks, with fine-tuned versions published on public repositories such as HuggingFace [35] and AdapterHub [29]. Model merging methods have emerged as a powerful way to leverage such checkpoints, aggregating them into one model with the combined capabilities of its constituents. Recently, *task arithmetic* [13] has emerged as a simple yet powerful paradigm for model merging. Rather than directly aggregating expert model parameters, e.g. by averaging them, *task vectors* are computed to isolate the difference between each expert's fine-tuned parameters and the pre-trained initialization. These task vectors are then aggregated, scaled, and added back to the pre-trained model parameters, often yielding significant performance gains. However, despite these advances, the merged model often falls short of each fine-tuned expert on its corresponding task, even while achieving better overall multi-task performance than any individual expert. This merging performance gap is generally attributed to task, or parameter, *interference* [36, 9, 22]: a phenomenon in which the accumulated parameter updates induced by training on different tasks can conflict during merging. Task interference is known to worsen as the number of models being merged increases [13, 36, 5, 22] and with longer fine-tuning duration [40, 12], motivating a wide range of task-arithmetic-based merging methods aimed at reducing negative interference [39, 9, 22, 37].

At first glance, low-communication distributed training methods such as DiLoCo and model merging methods appear to address very different problems. DiLoCo is motivated by the need to pre-train large models on massive datasets across multiple, poorly connected compute pools. In this setting, workers optimize the same objective, typically next-token prediction, on disjoint subsets of the same training dataset. By contrast, model merging is usually framed as a post-training technique: multiple checkpoints from a shared base model, fine-tuned on different data distributions and/or objectives, are combined into a single model intended to inherit their capabilities. Despite these differences, the two families of methods share a common underlying structure. In DiLoCo, the quantities communicated by workers at each outer optimization step are pseudo-gradients: parameter differences between the model at the previous global synchronization point and the locally updated replicas obtained after several inner optimization steps. From the perspective of model merging, these pseudo-gradients can be interpreted as task vectors, where the reference model is the checkpoint from the previous outer step and the "experts" are the local replica checkpoints after inner training. The main distinction is that DiLoCo repeats this train-and-aggregate procedure over many rounds, whereas standard model merging typically performs a single, often more substantial, round of specialization followed by one aggregation step. This connection also suggests that the performance degradation observed in DiLoCo as the number of workers or inner optimization steps increases is directly analogous to the negative task interference studied in the model merging literature. This parallel motivates viewing low-communication distributed training through the lens of model merging, and raises the question of whether merging methods designed to reduce interference can improve DiLoCo-style training.

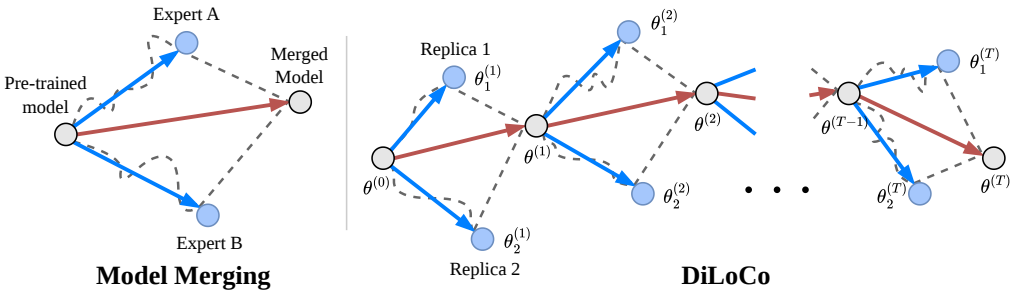

Figure 2: **DiLoCo as iterative model merging.** Dots represent model checkpoints, with blue dots denoting locally trained models: fine-tuned experts in model merging and worker replicas in DiLoCo respectively. Blue arrows are local updates—task vectors in merging and pseudo-gradients in DiLoCo—measuring displacement from the shared initialization. Red arrows denote aggregation: a one-shot merge on the left, and an outer-optimizer update on the right. Formally, with SGD, DiLoCo's outer step reduces to task-arithmetic merging (Equations (1) and (2)), applied iteratively.

In this work, we bridge these two closely related lines of research, which have thus far evolved independently. While DiLoCo has recently emerged as an effective approach for distributed pre-training across poorly connected compute pools, its global aggregation step remains simple, typically averaging the pseudo-gradients produced by each worker. In contrast, recent model merging methods propose more sophisticated aggregation mechanisms explicitly designed to reduce negative interference when combining task vectors from multiple independently fine-tuned models, including challenging settings with many experts or long fine-tuning durations. Building on the structural correspondence identified above, which we make explicit in Section 2.3, we investigate whether model merging techniques can serve as drop-in replacements for DiLoCo's pseudo-gradient aggregation step. Specifically, we ask whether such merging-inspired aggregation can better preserve performance as the number of workers and inner optimization steps increases, two regimes in which low-communication training is known to degrade. Our contributions are as follows:

- We establish a direct analogy between DiLoCo's pseudo-gradient aggregation step and task-arithmetic-based model merging (Section 2.3), showing that model merging methods can be interpreted as alternative aggregation rules for low-communication distributed training.

- Guided by this connection, we systematically benchmark state-of-the-art merging methods as drop-in aggregation mechanisms for DiLoCo-style training in Section 3, identifying Iso-C [22] as offering the strongest performance at reasonable computational cost, even surpassing DiLoCo despite not having a momentum mechanism itself (Figure 1 and Table 2).

- We propose **IsoLoCo** in Section 4, a practical adaptation of Iso-C to low-communication distributed training equipped with Nesterov momentum. Through extensive experiments across worker counts (up to $R = 128$), number of inner steps, and model sizes, we show that IsoLoCo consistently outperforms DiLoCo and substantially narrows the gap to data-parallel training.

- We develop a fast Newton–Schulz-based implementation of IsoLoCo that replaces SVD with iterative orthogonalization and an RMS proxy for the singular-value mean, yielding up to two orders of magnitude faster orthogonalization ($\approx$15% end-to-end speed-up) with negligible loss degradation ($\sim$0.45%).

- Following recent work that uses Muon as DiLoCo's inner optimizer (MuLoCo) [33], we show that IsoLoCo composes with a Muon inner optimizer to further improve worker-count scaling, achieving the lowest validation loss among all methods considered for $R \geq 32$, while IsoLoCo with an AdamW inner optimizer matches or surpasses MuLoCo at high worker counts, at a fraction of the orthogonalization cost (Section 4.4).

2 Background and Preliminaries

2.1 Distributed Low-Communication Training

The ideas behind DiLoCo can be traced back to federated learning, where training data is distributed across many consumer devices and cannot be centrally aggregated due to privacy or bandwidth constraints. Federated Averaging (FedAvg) [24] addresses this problem by broadcasting a global model to participating clients, letting each perform several local SGD steps, and aggregating the resulting updates through a weighted average. When each client takes exactly one step with the same batch size, FedAvg reduces to synchronous data-parallel SGD. Subsequent work has addressed FedAvg's shortcomings, such as difficulty of hyperparameter tuning and convergence issues, by incorporating adaptive server-side optimizers such as Adam [16] in the FedOpt framework [30].

DiLoCo [6] adapts this idea to large-scale language-model pretraining across R workers, or "islands" of compute. As shown in Algorithm 1, each worker receives the current global model and independently trains a local replica for H inner steps using an inner optimizer ($\mathrm{InnerOpt}$, often AdamW [21]). After the inner loop, each worker computes a pseudo-gradient capturing its local parameter changes, and these are averaged across workers. The global model is then updated with an outer optimizer ($\mathrm{OuterOpt}$, typically SGD with Nesterov momentum [26, 32]), applied to this averaged pseudo-gradient. Unlike standard FedOpt, DiLoCo retains inner optimizer states across outer rounds rather than resetting them. Since workers communicate only once every H steps, DiLoCo reduces global synchronization by roughly a factor of H relative to data-parallel, making it attractive for training across weakly connected or geographically distributed clusters.

Algorithm 1: DiLoCo

Require: Initial model $\theta^{(0)}$
Require: R workers, Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_R\}$
Require: Optimizers $\mathrm{InnerOpt}$ and $\mathrm{OuterOpt}$
for outer step $t = 1, \ldots, T$ **do**
  ▷ Inner optimization
  for worker $i = 1, \ldots, R$ **in parallel do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for inner step $h = 1, \ldots, H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      $\theta_i^{(t)} \leftarrow \mathrm{InnerOpt}\big(\theta_i^{(t)}, \nabla\mathcal{L}\big)$
    end for
  end for
  ▷ Average pseudo-gradients
  $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{i=1}^{R}\big(\theta^{(t-1)} - \theta_i^{(t)}\big)$
  ▷ Outer optimization
  $\theta^{(t)} \leftarrow \mathrm{OuterOpt}\big(\theta^{(t-1)}, \Delta^{(t)}\big)$
end for

Subsequent work has established that DiLoCo scales predictably with model size and can match or improve over data-parallel training under fixed compute budgets [4]. However, the same studies show that increasing the number of workers R degrades performance [4, 33]. Similarly, increasing the number of inner steps H leads to worse performance, as local replicas diverge further before aggregation, making the averaged pseudo-gradient less aligned with the update that tightly synchronized training would produce. This limitation is central to our work: DiLoCo's communication savings come at the cost of aggregation quality.

2.2 Model Merging and Task Arithmetic

Model merging is a means of combining multiple model checkpoints, fine-tuned from the same pre-trained model, into one with the goal of combining the constituent models' capabilities. It is often justified through the lens of *linear mode connectivity* [10, 7, 8]: models fine-tuned from a common pre-trained checkpoint tend to lie in the same basin of the loss landscape [27], meaning their parameters can be averaged without incurring high loss. However, naive parameter averaging often yields poor results in practice. A seminal improvement is *task arithmetic* (TA) [13], which operates on *task vectors* rather than raw parameters. Let θ_0 denote the pre-trained parameters and θ_t those of the model fine-tuned on task $t \in \mathcal{T}$. The task vector $\tau_t = \theta_t - \theta_0$ captures the parameter changes induced by fine-tuning, and models are merged as:

$$\mathrm{Merge}(\theta_0, \{\theta_t\}_{t\in\mathcal{T}}) = \theta_0 + \lambda \sum_{t\in\mathcal{T}} \tau_t, \tag{1}$$

where λ is a tunable hyper-parameter. Task-arithmetic based merging significantly improves merged model performance and has thus become the dominant paradigm in this area of research. Nevertheless, *negative parameter interference* remains a challenge, since the merged model often falls short of each

fine-tuned expert on its corresponding task. Such interference is exacerbated when the number of models being merged increases [13, 36, 5, 22] or when the fine-tuning stage is prolonged [40, 12].

Many methods have been proposed to reduce negative task interference during merging and narrow the gap between the merged model and the individual experts [13, 36, 39, 5, 9, 22, 37]. For example, some methods rely on sparsifying the task vectors to reduce direct parameter conflicts [36, 39, 5], while others use SVD to reduce conflicts in the spectral domain of task vectors [9, 22]. In this work we focus on five popular and state-of-the-art merging methods, representative of the two main mechanisms used to improve task-arithmetic-based merging: sparsity and orthogonalization. We provide a brief description of each considered method below.

TIES (TrIm, Elect Sign) merging was introduced in [36] as a means to reduce parameter interference. TIES prunes the low magnitude parameters of each task vector, then averages the remaining sparse task vectors based on sign alignment: in each parameter dimension, TIES only averages the parameters from each task vector that have the same sign as the weighted majority.

DARE (Drop And REscale) merging [39] randomly prunes a fraction of each task vector parameters and rescales the remaining ones based on the pruning fraction before combining them as in Eq. 1.

TSV (Task Singular Vector) merging [9] operates on the per-layer matrix form of task vectors, or task matrices, which were found empirically to be low-rank. TSV first computes a truncated SVD of each layer's task matrix to compress it into its principal singular directions, then decorrelates the singular vectors across tasks via a whitening transformation to reduce task interference before recombining.

Iso-C (Isotropic merging in Common subspace) **& Iso-CTS** [22] both operate on task matrices. Iso-C sums the task matrices, computes the SVD, and replaces the singular-value spectrum with its average, flattening high-energy directions to yield a more balanced common subspace. Iso-CTS further adds task-specific directions from the orthogonal complement to preserve information that may otherwise be lost during merging.

2.3 DiLoCo as Iterative Model Merging

We begin by highlighting the structural similarity between DiLoCo and task-arithmetic-based model merging. At outer step t, all DiLoCo workers are initialized from the shared parameters $\theta^{(t-1)}$ and independently perform local training, producing updated parameters $\{\theta_i^{(t)}\}_{i=1}^R$. DiLoCo then forms its outer update from the average of the resulting local updates, i.e., the displacement in parameter space between the shared initialization and the locally updated replicas. Defining the local update of worker i at outer step t as $\Delta_i^{(t)} := \theta_i^{(t)} - \theta^{(t-1)}$, with $\Delta^{(t)}$ denoting the average local update across workers, an SGD outer optimizer with learning rate η gives

$$\begin{aligned}\theta^{(t)} &= \theta^{(t-1)} + \eta\Delta^{(t)} \\ &= \theta^{(t-1)} + \eta\frac{1}{R}\sum_{i=1}^{R}\left(\theta_i^{(t)} - \theta^{(t-1)}\right). \end{aligned} \tag{2}$$

Comparing this update to Equation (1) reveals a direct correspondence: the shared parameters $\theta^{(t-1)}$ play the role of the pretrained model θ_0, each worker output $\theta_i^{(t)}$ plays the role of a task-specific fine-tuned model, the local updates $\Delta_i^{(t)}$ correspond to task vectors, and the number of workers corresponds to the number of tasks. The outer learning rate η then plays the role of the task-arithmetic scaling coefficient, up to the normalization by R.

Thus, with an SGD outer optimizer, DiLoCo's outer step can be viewed as an application of task-arithmetic-based model merging. This perspective raises a natural question: *can more sophisticated merging methods be substituted into the outer step to improve low-communication distributed training?* This is the question we investigate in the following section.

3 Can Model Merging Improve Pseudo-Gradient Aggregation in DiLoCo?

3.1 Experimental Set-up

Models, data and optimizers: We train Llama-style decoder-only transformer models of various sizes on the DCLM-Baseline [18] dataset. The data is tokenized with the Llama-2 [34] tokenizer and

the same training data order is used across training runs to minimize variability. We use a Chinchilla budget of 20x the number of training tokens relative to the number of model parameters [11] and a sequence length of 2048. We use a global batch size of 524k tokens for the 178M models and 2M tokens for the 512M and 1B models. We follow the convention of matching the global batch size for our DiLoCo experiments [6, 4], i.e. for any distributed training experiment with R workers, the inner batch size for each worker is set to global batch size divided by R.

For all DiLoCo experiments, we fix the number of warm-up steps to 10% of the entire training budget, we use AdamW as the inner optimizer with standard values of $\beta_1 = 0.9$ and $\beta_2 = 0.95$ and a fixed weight decay of 0.1. Unless otherwise specified, we use $R = 8$ workers, $H = 30$ inner steps, and SGD with Nesterov momentum [6, 4] for the outer optimizer.

Hyperparameter sweeps: For our distributed training experiments, we conduct an extensive sweep over outer momentum, outer learning rate, and inner learning rate at the 178M scale. At all other scales, we fix the outer hyperparameters (momentum and learning rate) to their optimal values at 178M and sweep only over the inner learning rate, since prior work has shown that the optimal outer hyperparameters remain approximately constant across model scales [4, 33]. The inner learning rate is swept over powers of $\sqrt{2}$, while the outer momentum is swept over values in $\{0.6, 0.7, 0.8, 0.9\}$. The outer learning rate is swept from 0.5 to 2 using increments of 0.1, following past works. However, since the merging inspired outer steps can differ substantially from the standard Nesterov SGD outer optimizer, we also find it necessary to sweep values greater than 2 using an approximately logarithmic spacing to preserve the granularity of the sweep across different scales. The training runs are evaluated on a held-out validation dataset consisting of a 100M token subset of the DCLM-Baseline dataset, and the best validation loss is reported. For the AdamW data-parallel baselines we sweep over the learning rate first with a fixed weight decay of 0.1 and then we sweep over the weight decay.

Code and compute: We use the PyTorch [28] implementation from [31], adapted to our setting. Most of our experiments were conducted on nodes with 4xH100 Nvidia GPUs, with some of the larger scale experiments (larger R and/or model size) conducted on nodes with 8xH200 Nvidia GPUs.

3.2 Model Merging for DiLoCo Pseudo-Gradient Aggregation

Motivated by the direct analogy between task-arithmetic-based model merging and the outer steps of DiLoCo with SGD identified in Section 2.3, we evaluate whether other model merging methods can improve DiLoCo by replacing its outer update. We use SGD as the outer optimizer because this is the setting in which the analogy holds, and to avoid the potentially confounding effects of outer momentum. We also include data-parallel (DP) AdamW and DiLoCo with SGD outer optimizer results for comparison. Results for $R = 8$ workers and $H = 30$ inner steps are presented in Table 1.

Table 1: Validation loss for DiLoCo SGD with merging-based aggregation.

Method	**Valid loss**
Baselines	
AdamW DP	2.881
DiLoCo SGD	2.990
DiLoCo SGD + Merging Agg.	
TIES	2.997
DARE	2.993
TSV-M	3.160
Iso-C	**2.929**
Iso-CTS	2.986

We first observe that the sparsity-based methods TIES and DARE achieve final validation losses close to DiLoCo SGD, with losses of 2.997 and 2.993 respectively. However, these results occur at the lowest sparsity value tested for both methods (0.1) and performance degrades progressively as sparsity increases. This suggests that their strong performance at low sparsity is primarily due to the resulting updates being similar to those of DiLoCo SGD, rather than to the benefits of sparsification itself. Thus, sparsity does not appear to provide a useful perspective for improving DiLoCo outer optimization in this setting.

For the orthogonalization-based methods, TSV-M performs poorly, reaching a validation loss of 3.160, approximately 5.6% higher than the DiLoCo SGD baseline. It is also among the most computationally expensive methods evaluated, requiring $T+2$ singular value decompositions for every parameter matrix in the model: one for each worker for the low-rank approximation and two additional global. While this cost may be acceptable in merging settings, where the method is only applied once after training, it becomes prohibitive in the DiLoCo setting, where the procedure is applied at every outer optimization step. We therefore exclude TSV-M from further analysis.

The related Iso-C and Iso-CTS methods both outperform the DiLoCo SGD baseline, achieving validation losses of 2.929 and 2.986 respectively. However, Iso-CTS is substantially more computationally

expensive, requiring $T + 1$ singular value decompositions and therefore incurring a cost on the order of the number of workers like TSV-M. Moreover, Iso-CTS performs worse than Iso-C in our setting, and prior work suggests that the difference between the two is small even for merging [22]. While preserving task-specific information may be marginally beneficial in model merging, where models are trained for longer and often on substantially different tasks, it appears less appropriate for DiLoCo: replicas are trained only for short inner loops, on subsets of the same dataset, and under the same objective. In this setting, the additional mechanisms introduced by Iso-CTS appear unnecessary and may even harm performance. We therefore also exclude Iso-CTS from the remainder of our analysis.

Our benchmarking of model merging methods as outer optimizers for DiLoCo therefore leaves Iso-C as the strongest candidate, improving over the DiLoCo SGD baseline by approximately 2% with modest computational overhead: a single SVD per two-dimensional parameter matrix, rather than $O(T)$ decompositions as in TSV-M and Iso-CTS. As we will show later, even this cost can be significantly reduced. We therefore adopt Iso-C as the foundation for the remainder of our analysis.

4 IsoLoCo: Isotropic Distributed Low-Communication Training

Building on Iso-C, we introduce IsoLoCo (**Iso**tropic distributed **Lo**w-**Co**mmunication) training, which augments the isotropic correction with Nesterov momentum. This is motivated by [6], who identified Nesterov momentum as highly beneficial for FedOpt algorithms in distributed low-communication LLM training; we hypothesize it can also improve Iso-C.

The algorithm is presented in Algorithm 2. At each outer step t, each worker i provides a pseudo-gradient $\Delta_i^{(t)} := \theta_i^{(t)} - \theta^{(t-1)}$, representing its local progress relative to the previous synchronized global parameters $\theta^{(t-1)}$. These are averaged to form $\Delta^{(t)}$. For weight matrices (parameters with exactly two dimensions), we apply the isotropic correction: the SVD of $\Delta^{(t)}$ is computed, its singular values replaced by their mean $\bar{\sigma}$, and the update reconstructed as $\bar{\sigma}UV^\top$, projecting onto a nearby isotropic operator that regularizes the spectrum while preserving subspace structure. Non-matrix parameters (e.g. normalization layers) are left unchanged. The corrected pseudo-gradient is then incorporated into a Nesterov momentum scheme: the buffer $B^{(t)}$ accumulates past corrected pseudo-gradients, and the effective update $P^{(t)}$ applies the characteristic look-ahead.

Algorithm 2: IsoLoCo

Require: Initial global model $\theta^{(0)}$
Require: Outer learning rate η_{out}, momentum μ
Initialize momentum buffer $B^{(0)} \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  Complete local optimization from $\theta^{(t-1)}$
  Collect pseudo-gradients $\{\Delta_i^{(t)}\}_{i=1}^R$
  Compute mean $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{i=1}^R \Delta_i^{(t)}$
  for parameter tensor ℓ **do**
    if $\dim(\Delta_\ell^{(t)}) = 2$ **then**
      $U\Sigma V^\top \leftarrow \mathrm{SVD}(\Delta_\ell^{(t)})$
      $\boldsymbol{\sigma} \leftarrow \mathrm{Diag}(\Sigma)$
      $\bar{\sigma} \leftarrow \frac{1}{r}\sum_{k=1}^r \boldsymbol{\sigma}_k$
      $\Delta_\ell^{(t)} \leftarrow \bar{\sigma}UV^\top$
    end if
  end for
  $B^{(t)} \leftarrow \mu B^{(t-1)} + \Delta^{(t)}$
  $P^{(t)} \leftarrow \Delta^{(t)} + \mu B^{(t)}$
  $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_{\text{out}} P^{(t)}$
end for

4.1 IsoLoCo Outperforms DiLoCo at High Worker Counts

We now review our low-communication LLM pre-training results, where the number of workers we train on grows from $R = 1$ to 128 and the synchronization interval is fixed at $H = 30$. To the best of our knowledge, we scaled the number of workers well beyond the largest settings studied in past works $R \approx 16$–20 [33, 19, 14]. Our results are presented in Tables 2 and 3 and Figure 1.

Table 2 presents the final validation loss of DiLoCo, DiLoCo SGD, DiLoCo SGD + Iso-C, and IsoLoCo on 178M models and, for DiLoCo and IsoLoCo, on 512M models. Surprisingly, even the naive combination of DiLoCo SGD with Iso-C merging-based aggregation outperforms DiLoCo at 178M scale across all worker counts, despite lacking a momentum mechanism. IsoLoCo further widens this gap, outperforming DiLoCo across both model scales and confirming the added benefit of momentum. At 178M scale, we find that the improvement of 0.027 loss at $R = 1$ ($\sim 0.9\%$) grows to 0.195 at $R = 128$ ($\sim 5.8\%$), showing IsoLoCo's substantial improvements over DiLoCo at large worker counts. At the 512M scale, IsoLoCo's improvement over DiLoCo is just as strong, growing

Table 2: **Validation loss across numbers of workers R and model sizes.** The AdamW DP baseline reached 2.881 and 2.633 final loss at 178M and 512M scales, respectively. We observe that IsoLoCo consistently improves over DiLoCo and has much stronger performance at large worker counts.

Model	Method	$R=1$	$R=2$	$R=4$	$R=8$	$R=16$	$R=32$	$R=64$	$R=128$
178M	DiLoCo SGD	2.887	2.905	2.937	2.990	3.064	3.178	3.357	3.601
	DiLoCo	2.869	2.882	2.911	2.950	3.005	3.077	3.195	3.355
	DiLoCo SGD + Iso-C	2.855	2.872	2.891	2.929	2.992	3.070	3.162	3.280
	IsoLoCo (ours)	**2.842**	**2.852**	**2.864**	**2.889**	**2.930**	**2.992**	**3.062**	**3.160**
512M	DiLoCo	2.630	2.639	2.653	2.675	2.706	2.767	2.863	–
	IsoLoCo (ours)	**2.589**	**2.593**	**2.599**	**2.610**	**2.628**	**2.676**	**2.716**	–

from 0.041 (1.6%) at $R=1$ to 0.147 (5.1%) at $R=64$. The $R=128$ results for the 512M models are omitted due to out-of-memory issues at that scale.

In Table 3 we also report results for 1B-scale models with $R=8$ and $H=30$. IsoLoCo continues to outperform DiLoCo at this scale (2.421 vs. 2.450), though the relative improvement is more modest than at smaller scales: 1.2% at 1B, compared to 2.1% and 2.4% at 178M and 512M (all with $R=8$). Two effects help explain this. First, DiLoCo's worker scaling is known to improve with model size [4, 33], allowing it to catch up to IsoLoCo at the larger scales. Second, larger models reach lower validation loss overall, leaving less headroom for algorithmic gains. Taken together, these suggest that IsoLoCo should continue to outperform DiLoCo as model size increases, albeit with smaller relative improvements at larger scales.

Table 3: Validation loss for DiLoCo and IsoLoCo for 1B models, $R=8$.

Method	Valid loss
DiLoCo	2.450
IsoLoCo (ours)	**2.421**

Figure 1 reports the performance of DiLoCo, DiLoCo SGD + Iso-C, and IsoLoCo at 178M scale normalized relative to an AdamW DP baseline. Specifically, we report the percentage $100 \cdot (\frac{\text{Method}}{\text{AdamW DP}} - 1)$, i.e. the relative increase in loss over AdamW DP. Since communication-intensive data-parallel training is the de facto gold standard and low communication methods are known to degrade in performance at larger worker counts [33, 4], this metric allows direct inspection of the performance degradation of different methods relative to this ideal. We observe that relative to the AdamW DP (Loss=2.881) all low communication methods degrade as the number of workers increases. However, IsoLoCo strictly improves over DiLoCo at all worker counts, reaching only a 9.7% degradation at $R=128$ vs. DiLoCo's 16.5%. Moreover, IsoLoCo outperforms AdamW DP at $R \leq 4$. In summary, we find that IsoLoCo yields strict improvements over DiLoCo in all settings.

4.2 Increasing the Number of Inner Steps

We now evaluate the performance of our method as the synchronization interval is increased from $H=30$ to 240. The number of workers is fixed to $R=8$ here. Table 4 reports the final loss of DiLoCo and IsoLoCo at each synchronization interval $H \in \{30, 60, 120, 240\}$. Both methods were extensively tuned in each case. We observe that IsoLoCo consistently improves over DiLoCo across all numbers of inner steps. These results demonstrate that IsoLoCo is compatible with long synchronization intervals, which is essential for training under heavy bandwidth constraints.

Table 4: **Validation loss as a function of synchronization interval (H) for DiLoCo and IsoLoCo.** We observe that IsoLoCo consistently improves over DiLoCo across all synchronization intervals.

Method	$H=30$	$H=60$	$H=120$	$H=240$
DiLoCo	2.950	2.957	2.976	3.008
IsoLoCo (ours)	**2.889**	**2.907**	**2.944**	**2.991**

4.3 Accelerating IsoLoCo with Newton-Schulz Iterations

Despite being significantly less computationally expensive than some of the other merging methods considered, Iso-C still relies on SVD to compute the singular values and vectors of the gradients, which can be slow on GPUs. We therefore explore whether IsoLoCo can be accelerated using the iterative Newton-Schulz [3, 17] method, which is already used in the recently popularized Muon

optimizer [15, 20]. We discuss the relationship between IsoLoCo and Muon in more detail in Appendix C. Let $M \in \mathbb{R}^{n\times m}$ be a matrix with singular value decomposition $M = USV^\top$. The Newton-Schulz (NS) method uses matrix multiplications to approximately orthogonalize M, i.e. $\mathrm{NS}(M) \approx UV^\top$. However, using Newton-Schulz to orthogonalize the gradients does not yield the individual singular values, which are needed to compute their mean in IsoLoCo. Instead, we propose using the root mean square of the singular values as a cheap proxy, since it can be computed directly from the gradient matrix before orthogonalization:

$$\mathrm{RMS}(\sigma) = \frac{\|M\|_F}{\sqrt{r}} = \sqrt{\frac{1}{r}\sum_{i=1}^{r}\sigma_i^2}.$$

Note that in general $\mathrm{RMS}(\sigma) \geq \mathrm{mean}(\sigma)$, with equality only when all singular values are identical. Despite this difference, we find that the RMS is a good replacement for the mean. In our experiments with 178M models, $R = 8$, and $H = 30$, the fast implementation reaches a validation loss of 2.902, still significantly outperforming DiLoCo (2.950) and trailing the SVD implementation (2.889) by only $\sim$0.45%. Furthermore, both versions share the same optimal hyperparameters.

Time complexity analysis. Both SVD and Newton-Schulz have a time complexity of $O(\min(m^2n, mn^2))$, with Newton-Schulz also depending on the number of iterations; however, that number is small (we use the standard 5-step iteration from the literature [15, 20] with the polynomial coefficients from [1]). In practice, Newton-Schulz is faster because it relies solely on general matrix-matrix multiplications, which GPUs are heavily optimized for and which can exploit lower-precision arithmetic. At the 178M model scale with $R = 8$, we found the Newton–Schulz orthogonalization step to be **up to two orders of magnitude faster** than SVD for the matrix sizes used in our models, resulting in a roughly 21% speed-up in wall-clock time over the entire training duration, putting IsoLoCo with NS within 0.6% of DiLoCo's median wall-clock time.

4.4 IsoLoCo improves MuLoCo's worker-count scaling

The Muon optimizer [15] has recently attracted considerable attention for LLM pre-training [20, 2], as it improves on both the performance and efficiency of the long-standing default optimizer, AdamW [21]. Closest to our setting, [33] use Muon as DiLoCo's *inner* optimizer and show that the resulting method, MuLoCo, scales better with worker count, tolerates tighter communication constraints, and admits larger critical batch sizes than standard DiLoCo. Since IsoLoCo modifies only the *outer* step, it is orthogonal to the choice of inner optimizer. We therefore ask whether the two improvements compose, pairing IsoLoCo's outer step with a Muon inner optimizer.

Figure 3 compares DiLoCo and IsoLoCo under both AdamW [21] and Muon [15] inner optimizers as the worker count is varied from $R = 8$ to $R = 128$ in a FLOP-matched setting; following [33], we denote DiLoCo with a Muon inner optimizer as MuLoCo.

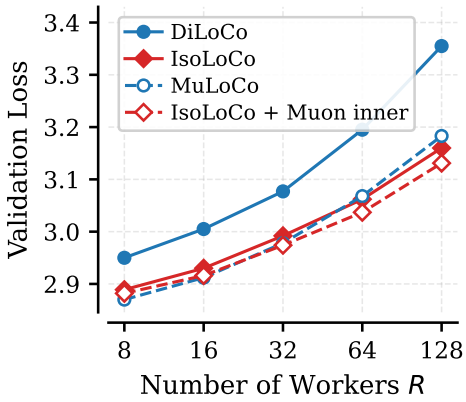

Figure 3: **IsoLoCo further improves MuLoCo's worker-count scaling.** Final validation loss is shown versus worker count for a 178M LLaMa-2 model in a FLOP-matched setting.

The two techniques improve DiLoCo through the same underlying mechanism—flattening the spectrum of the updates—but at different stages: IsoLoCo replaces the singular values of the outer update with their mean, while Muon drives the singular values of the inner updates toward one. This shared mechanism explains both why combining them helps and why the effect is not additive. Each operation improves over DiLoCo on its own, and because they act at different stages they contribute complementary gains: IsoLoCo's isotropic scaling improves the aggregation of worker gradients, yielding its largest advantages at high worker counts, while Muon's inner steps improve convergence. Combining them therefore helps, and increasingly so as workers scale. But since both operations flatten the spectrum, stacking them does not double the benefit. Because the outer step is applied only once every H inner steps, IsoLoCo orthogonalizes far less frequently than MuLoCo, which does so at every inner step. IsoLoCo is therefore considerably cheaper while matching MuLoCo at low worker counts and surpassing it at high ones.

This is borne out by the numbers. At $R = 8$, IsoLoCo, MuLoCo, and IsoLoCo with a Muon inner optimizer all fall within $\sim 0.7\%$ of one another, and all sit more than $\sim 2\%$ below DiLoCo: at low worker counts the two operations do largely the same job, so little is gained by applying both. As the worker count grows, however, the combination of IsoLoCo and MuLoCo increasingly separates from either method alone. For $R \geq 32$ it attains the lowest validation loss among all methods considered, with its improvement over MuLoCo growing from 0.13% at $R = 32$ to 1.63% at $R = 128$, and its improvement over IsoLoCo with an AdamW inner optimizer growing from 0.60% to 0.92% over the same range. This is consistent with the outer and inner interventions addressing distinct bottlenecks, namely gradient aggregation and convergence respectively, both becoming more pronounced at scale.

4.5 Ablations and Analysis

Momentum and orthogonalization target. We compare Nesterov momentum against standard (non-Nesterov) momentum and evaluate whether orthogonalization should be applied to the gradients $\Delta^{(t)}$ (as in IsoLoCo) or to the final update matrix $P^{(t)}$. For the momentum experiments, we additionally sweep over the outer momentum coefficient, as the non-Nesterov variant requires lower values for this hyperparameter. We find that all design choices have a modest effect: non-Nesterov momentum yields slightly worse results ($\sim 0.2\%$) than the Nesterov variant, and applying orthogonalization to $P^{(t)}$ rather than the gradients also degrades performance slightly. We additionally tried excluding the two-dimensional embedding and head parameters from orthogonalization but observed no improvement.

Directional analysis of singular value equalization. Since IsoLoCo exposes the gradients' singular values, we can isolate the source of its gains: does performance improve primarily from clipping high singular values down to the mean, or from boosting low singular values up to it? We implement two variants: CLIP_HIGH, which sets only above-mean singular values to the mean, and CLIP_LOW, which does the same for below-mean values, leaving the remaining singular values unchanged. Both variants improve over the DiLoCo baseline (2.950), with CLIP_LOW reaching 2.931 and CLIP_HIGH reaching 2.894. Notably, CLIP_HIGH nearly matches the full IsoLoCo result (2.889), while CLIP_LOW closes only a small fraction of the gap. **We therefore conclude that IsoLoCo's gains are primarily driven by reducing the dominance of a few high-energy directions in the gradient spectrum.**

5 Conclusion and Limitations

This work investigates model merging as a mechanism for improving gradient aggregation in DiLoCo-style low-communication training, providing what is, to our knowledge, the first thorough study of model merging methods in the context of distributed optimization. Our analysis is grounded in the direct analogy between task-arithmetic-based model merging and DiLoCo with SGD outer optimizer (Section 2.3). Evaluating merging methods in this setting allows us to identify Iso-C as a particularly effective replacement for pseudo-gradient averaging and propose IsoLoCo, a momentum-equipped variant for distributed optimization. IsoLoCo consistently outperforms DiLoCo across model sizes and synchronization intervals, with gains that become more pronounced as the number of workers increases. We further introduce a fast implementation based on the iterative Newton–Schulz method that closely matches the SVD version while yielding substantial wall-clock improvements.

More broadly, our results suggest that model merging is a valuable source of inspiration for distributed training research: merging-inspired gradient aggregation can help low-communication training scale to larger numbers of poorly interconnected workers while reducing the performance gap to AdamW DP typically observed with standard DiLoCo. Important future directions include evaluating IsoLoCo under strong compression, combining it with low-communication pipelining methods, studying the single-worker case more extensively, and developing a theoretical understanding of why merging-inspired distributed optimizers, IsoLoCo in particular, improve results.

Limitations. While we have made our best effort to provide a thorough scientific study and taken the necessary measures to ensure our results are sound, limitations naturally arise from the computational resources available in academia. In no particular order, our empirical results do not evaluate models beyond 1B scale, architectures beyond a dense transformer, or tasks beyond language modeling. On the theoretical side, we do not prove convergence for optimizers employing merging outer steps.

Broader Impact. This work proposes improvements to distributed optimization for neural network training, which could help make large-scale training more accessible to researchers with limited or heterogeneous infrastructure. We do not foresee specific negative societal consequences beyond those generally associated with advances in language modeling.

Acknowledgments

This work was partially funded by the NSERC CGS D 569345 - 2022 scholarship [S.H.]; FRQNT Doctoral (B2X) scholarship [B.T.]; FRQNT New Scholar grant [E.B.]; FRQNT-NSERC grant 2023-NOVA-329125 [E.B. & G.W.]; Canada CIFAR AI Chair, NSF DMS grant 2327211 and NSERC Discovery grant 03267 [G.W.]. This work is also supported by resources from Compute Canada and Calcul Quebec. The content is solely the responsibility of the authors and does not necessarily represent the views of the funding agencies.

References

[1] K. Ahn, B. Xu, N. Abreu, Y. Fan, G. Magakyan, P. Sharma, Z. Zhan, and J. Langford. Dion: Distributed orthonormalized updates, 2025.

[2] E. AI, :, I. Shah, A. M. Polloreno, K. Stratos, P. Monk, A. Chaluvaraju, A. Hojel, A. Ma, A. Thomas, A. Tanwer, D. J. Shah, K. Nguyen, K. Smith, M. Callahan, M. Pust, M. Parmar, P. Rushton, P. Mazarakis, R. Kapila, S. Srivastava, S. Singla, T. Romanski, Y. Vanjani, and A. Vaswani. Practical efficiency of muon for pretraining, 2025.

[3] Å. Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971.

[4] Z. Charles, G. Teston, L. M. Dery, J. K. Rush, N. Fallen, Z. Garrett, A. Szlam, and A. Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[5] M. Davari and E. Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXXV*, page 270–287, Berlin, Heidelberg, 2024. Springer-Verlag.

[6] A. Douillard, Q. Feng, A. A. Rusu, R. Chhaparia, Y. Donchev, A. Kuncoro, M. Ranzato, A. Szlam, and J. Shen. Diloco: Distributed low-communication training of language models, 2024.

[7] F. Draxler, K. Veschgini, M. Salmhofer, and F. Hamprecht. Essentially no barriers in neural network energy landscape. In J. Dy and A. Krause, editors, *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1309–1318. PMLR, 10–15 Jul 2018.

[8] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In H. D. III and A. Singh, editors, *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR, 13–18 Jul 2020.

[9] A. A. Gargiulo, D. Crisostomi, M. S. Bucarelli, S. Scardapane, F. Silvestri, and E. Rodolà. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18695–18705, June 2025.

[10] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31. Curran Associates, Inc., 2018.

[11] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models, 2022.

[12] S. Horoi, G. Wolf, E. Belilovsky, and G. K. Dziugaite. Less is more: Undertraining experts improves model upcycling. *arXiv preprint arXiv:2506.14126*, 2025.

[13] G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations (ICLR)*, 2023.

[14] S. Jaghouar, J. M. Ong, M. Basra, F. Obeid, J. Straube, M. Keiblinger, E. Bakouch, L. Atkins, M. Panahi, C. Goddard, M. Ryabinin, and J. Hagemann. INTELLECT-1 technical report. *CoRR*, abs/2412.01152, 2024.

[15] K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024. Accessed: 2026-04-27.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[17] Z. Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 7(3):386–389, 1970.

[18] J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, S. Garg, R. Xin, N. Muennighoff, R. Heckel, J. Mercat, M. Chen, S. Gururangan, M. Wortsman, A. Albalak, Y. Bitton, M. Nezhurina, A. Abbas, C.-Y. Hsieh, D. Ghosh, J. Gardner, M. Kilian, H. Zhang, R. Shao, S. Pratt, S. Sanyal, G. Ilharco, G. Daras, K. Marathe, A. Gokaslan, J. Zhang, K. Chandu, T. Nguyen, I. Vasiljevic, S. Kakade, S. Song, S. Sanghavi, F. Faghri, S. Oh, L. Zettlemoyer, K. Lo, A. El-Nouby, H. Pouransari, A. Toshev, S. Wang, D. Groeneveld, L. Soldaini, P. W. Koh, J. Jitsev, T. Kollar, A. G. Dimakis, Y. Carmon, A. Dave, L. Schmidt, and V. Shankar. Datacomp-lm: In search of the next generation of training sets for language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 14200–14282. Curran Associates, Inc., 2024.

[19] J. Lidin, A. Sarfi, E. Miahi, Q. Anthony, S. Chauhan, E. Pappas, B. Thérien, E. Belilovsky, and S. Dare. Covenant-72b: Pre-training a 72b llm with trustless peers over-the-internet, 2026.

[20] J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, Y. Chen, H. Zheng, Y. Liu, S. Liu, B. Yin, W. He, H. Zhu, Y. Wang, J. Wang, M. Dong, Z. Zhang, Y. Kang, H. Zhang, X. Xu, Y. Zhang, Y. Wu, X. Zhou, and Z. Yang. Muon is scalable for llm training, 2025.

[21] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[22] D. Marczak, S. Magistri, S. Cygert, B. Twardowski, A. D. Bagdanov, and J. van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In *Forty-second International Conference on Machine Learning*, 2025.

[23] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018.

[24] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[25] W. Merrill, S. Arora, D. Groeneveld, and H. Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 116936–116959. Curran Associates, Inc., 2025.

[26] Y. Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[27] B. Neyshabur, H. Sedghi, and C. Zhang. What is being transferred in transfer learning? In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 512–523. Curran Associates, Inc., 2020.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc., 2019.

[29] J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, and I. Gurevych. AdapterHub: A framework for adapting transformers. In Q. Liu and D. Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online, Oct. 2020. Association for Computational Linguistics.

[30] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[31] A. Sarfi, B. Thérien, J. Lidin, and E. Belilovsky. Communication efficient llm pre-training with sparseloco. `https://arxiv.org/pdf/2508.15706`, 2025.

[32] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[33] B. Thérien, X. Huang, A. Defazio, I. Rish, and E. Belilovsky. Muloco: Muon is a practical inner optimizer for diloco, 2026.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[35] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[36] P. Yadav, D. Tam, L. Choshen, C. A. Raffel, and M. Bansal. Ties-merging: Resolving interference when merging models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 7093–7115. Curran Associates, Inc., 2023.

[37] E. Yang, L. Shen, G. Guo, X. Wang, X. Cao, J. Zhang, and D. Tao. Model merging in LLMs, MLLMs, and beyond: Methods, theories, applications, and opportunities. *ACM Comput. Surv.*, 58(8), Feb. 2026.

[38] G. Yang, J. B. Simon, and J. Bernstein. A spectral condition for feature learning, 2024.

[39] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 57755–57775. PMLR, 21–27 Jul 2024.

[40] L. Zhou, D. Solombrino, D. Crisostomi, M. S. Bucarelli, G. A. D'Inverno, F. Silvestri, and E. Rodolà. On task vectors and gradients. In *UniReps: 3rd Edition of the Workshop on Unifying Representations in Neural Models*, 2025.

A Optimal Hyperparameters

We report the optimal hyperparameters used for each main-text result table: inner learning rate η_{in}, outer learning rate η_{out}, and outer momentum μ. For the AdamW DP baselines we found optimal learning rates of 0.00195 and 0.00276; weight decays of 0.15 and 0.125 for the 178M and 512M models, respectively.

Table 5: Optimal hyperparameters for Table 1.

Method	η_{in}	η_{out}
DiLoCo SGD	0.00138	1.6
TIES	0.00276	0.7
DARE	0.00276	1.5
TSV-M	0.00276	0.6
Iso-C	0.00391	2.1
Iso-CTS	0.00391	3.1

Table 6: Optimal hyperparameters for Table 2 at 178M scale.

Method	**Hyperparameter**	$R=1$	$R=2$	$R=4$	$R=8$	$R=16$	$R=32$	$R=64$	$R=128$
DiLoCo SGD	η_{in}	0.00195	0.00195	0.00276	0.00138	0.00195	0.00138	0.00138	0.000977
	η_{out}	1.2	1.4	1.4	1.6	1.5	1.6	1.6	1.6
DiLoCo	η_{in}	0.00138	0.00138	0.00138	0.00138	0.00138	0.00195	0.00276	0.00195
	η_{out}	0.5	0.6	0.6	0.7	0.8	1	0.9	1
	μ	0.8	0.8	0.8	0.9	0.9	0.9	0.9	0.9
DiLoCo SGD + Iso-C	η_{in}	0.00391	0.00391	0.00391	0.00391	0.00138	0.000488	0.000244	0.000122
	η_{out}	1.7	1.7	2.1	2.1	4.6	9	15.9	25.5
IsoLoCo	η_{in}	0.00195	0.00138	0.00138	0.00138	0.00195	0.000345	0.000244	0.000122
	η_{out}	1.6	1.7	1.8	1.8	1.8	4.6	5.6	9.8
	μ	0.7	0.8	0.8	0.8	0.8	0.9	0.9	0.9

Table 7: Optimal hyperparameters for Table 2 at 512M scale.

Method	**Hyperparameter**	$R=1$	$R=2$	$R=4$	$R=8$	$R=16$	$R=32$	$R=64$
DiLoCo	η_{in}	0.00195	0.00138	0.00195	0.000691	0.000691	0.000488	0.000488
	η_{out}	0.5	0.6	0.6	0.7	0.8	1	0.9
	μ	0.8	0.8	0.8	0.9	0.9	0.9	0.9
IsoLoCo	η_{in}	0.00195	0.00138	0.00195	0.00138	0.00195	0.000345	0.000345
	η_{out}	1.6	1.7	1.8	1.8	1.8	4.6	5.6
	μ	0.7	0.8	0.8	0.8	0.8	0.9	0.9

Table 8: Optimal hyperparameters for Table 4.

Method	**Hyperparameter**	$H=30$	$H=60$	$H=120$	$H=240$
DiLoCo	η_{in}	0.00138	0.00138	0.00195	0.00195
	η_{out}	0.7	0.9	1	1.1
	μ	0.9	0.8	0.8	0.7
IsoLoCo	η_{in}	0.00138	0.00138	0.000977	0.00138
	η_{out}	1.8	1.7	1.7	1.7
	μ	0.8	0.8	0.8	0.7

Table 9: Optimal hyperparameters for Figure 3 – Muon as inner optimizer. For DiLoCo and IsoLoCo optimal hyperparameters with AdamW inner optimizer see Table 6.

Method	Hyperparameter	$R = 8$	$R = 16$	$R = 32$	$R = 64$	$R = 128$
MuLoCo (DiLoCo + Muon inner)	η_{in}	0.00552	0.00391	0.00276	0.00276	0.00391
	η_{out}	0.7	1.0	0.9	1.1	1.0
	μ	0.8	0.8	0.9	0.9	0.9
IsoCLoCo + Muon inner	η_{in}	0.00391	0.00276	0.00138	0.00195	0.00195
	η_{out}	1.2	1.6	1.9	1.8	2.1
	μ	0.8	0.8	0.9	0.9	0.9

B IsoLoCo Is Less Sensitive to the Outer Learning Rate than DiLoCo

We observe that IsoLoCo favors higher outer learning rates than DiLoCo at large worker counts, with optimal values of η_{out} exceeding 2 for $R \geq 32$. Importantly, as noted in Section 3.1, our learning rate sweeps use logarithmically spaced grids, preserving the relative granularity of the search across different scales. To further investigate sensitivity to the outer learning rate, we conduct an additional experiment with 178M models and $R = 32$ workers. Specifically, we fix η_{out} to different multiples $\{\frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1, \frac{3}{2}, 2, 3\}$ of the optimal value η^*_{out} reported in Table 6 for each method. For each fixed outer learning rate, we sweep the inner learning rate and outer momentum and report the lowest validation loss.

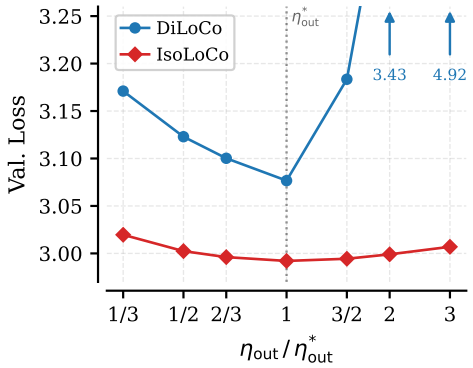

Figure 4: **IsoLoCo is less sensitive to the outer learning rate than DiLoCo.** Final validation loss versus the ratio $\eta_{\text{out}}/\eta^*_{\text{out}}$, where η^*_{out} denotes the optimal outer learning rate for each method. For each fixed outer learning rate, the inner learning rate and outer momentum are swept.

The results, shown in Figure 4, demonstrate that IsoLoCo is substantially less sensitive to the outer learning rate than DiLoCo. Across outer learning rates ranging from $\frac{1}{3}\times$ to $3\times$ their respective optimal values, IsoLoCo's validation loss increases by less than 1% relative to its optimum, whereas DiLoCo's increases by more than 3% at $\frac{1}{3}\times$ and nearly 60% at $3\times$. We hypothesize that IsoLoCo's ability to accommodate larger outer learning rates stems from its orthogonalization procedure, consistent with observations that Muon permits larger learning rates than AdamW in conventional data-parallel training [15, 20].

C Comparing IsoLoCo to Muon

IsoLoCo bears some similarities with the Muon optimizer, which has been recently proposed as an effective optimizer for 2D parameters in LLMs [15, 20]. In this section, we discuss the similarities and differences between the two.

Muon approximately orthogonalizes the parameter gradients through the use of the Newton-Schulz iterative method described in Section 4.3. Muon then scale the gradients by $\sqrt{\text{fan_out}/\text{fan_in}}$ so that the effective magnitude of the gradients is consistent across layers of different sizes [38]. We will call this normalization here "spectral".

Algorithm 3: Muon

Require: Learning rate η, momentum μ
1: Initialize momentum buffer $B_0 \leftarrow 0$
2: **for** $t = 1, \ldots, T$ **do**
3: Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
4: $B_t \leftarrow \mu B_{t-1} + G_t$
5: $O_t \leftarrow \mathrm{NewtonSchulz}(B_t)$
6: $O_t \leftarrow \sqrt{\text{fan_out}/\text{fan_in}} \cdot O_t$
7: Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
8: **end for**
9: **return** θ_T

The main difference between IsoLoCo is that IsoLoCo actually computes the SVD of the gradient matrices, giving us access to the singular values and allowing to set them all to their mean. On the other hand, Muon performs Newton-Shulz iterations to approximately set all singular values to 1 and then normalizes then by $\sqrt{\text{fan_out}/\text{fan_in}}$. Since we also use Newton–Schulz in the fast implementation of IsoLoCo, the main difference becomes the scaling: IsoLoCo scales the orthogonalized gradient matrices by the mean/RMS of their singular values while Muon scales by $\sqrt{\text{fan_out}/\text{fan_in}}$.

We isolate the impact of the orthogonalization (SVD vs. Newton–Shulz) and the scaling (mean vs spectral) on IsoLoCo by implementing these variants and keeping all other factors constant. We report results for our 178M models with $R = 8$ and $H = 30$ in Table 10. The results with SVD orthogonalization and mean scaling (IsoLoCo) and with Newton–Schulz orthogonalization and RMS scaling (fast implementation of IsoLoCo) were already presented in the main text, with both variants performing well. On the other hand, Newton–Schulz orthogonalization with $\sqrt{\text{fan_out}/\text{fan_in}}$ scaling, which roughly represents Muon as a DiLoCo outer optimizer, is significantly worse than IsoLoCo, with 3.4% higher loss.

Table 10: Validation loss for scaling choices under different orthogonalization methods.

	Orthog. Method	
	SVD	**Newton-Schulz**
Mean/RMS scale	2.898	2.902
$\sqrt{\frac{\texttt{fan_out}}{\texttt{fan_in}}}$ scale	–	3.005

Other notable differences between IsoLoCo and Muon are the fact that Muon uses standard momentum while IsoLoCo uses Nesterov. Also, Muon applies the Newton–Schulz-based orthogonalization to the final update matriecs B_t while IsoLoCo applies the orthogonalization to the gradients directly. Both of these variations were found to perform worse than our IsoLoCo design (see Section 4.5). Another difference is in how these two methods treat 1D parameters, IsoLoCo simply averages them and applies momentum (similar to SGD + Nesterov optimizer) while Muon typically optimizes those with another optimizer such as AdamW. Lastly, it is common in Muon implementations to exclude the 2D embedding and head parameters from the orthogonalization step, we did not find this to significantly affect performance.

D How IsoLoCo Mitigates Inter-Worker Conflicts

Although IsoLoCo applies isotropization only after averaging the workers' pseudo-gradients, the resulting spectrum implicitly reflects interactions between workers. Writing $\Delta = \frac{1}{R}\sum_{i=1}^{R}\Delta_i$, we have

$$\Delta\Delta^\top = \frac{1}{R^2}\left[\sum_i \Delta_i\Delta_i^\top + \sum_{i\neq j}\Delta_i\Delta_j^\top\right].$$

Since $\Delta\Delta^\top = U\Sigma^2U^\top$ determines the singular values Σ that Iso-C equalizes, the cross terms $\Delta_i\Delta_j^\top$ directly influence the spectrum on which Iso-C operates. These terms capture interactions between worker updates, including how their contributions reinforce or cancel along different directions. In particular,

$$\mathrm{Iso}\Big(\sum_i \Delta_i\Big) \neq \sum_i \mathrm{Iso}(\Delta_i)$$

in general. Thus, although Iso-C operates only on the averaged update, its nonlinear spectral transformation implicitly responds to the interactions between workers encoded in that average. While Iso-C does not explicitly identify or resolve pairwise conflicts, it can mitigate their effects by rebalancing the spectrum of the aggregated update: directions along which worker contributions reinforce one another may become disproportionately dominant singular modes, and Iso-C reduces their relative influence. This interpretation is supported by our experiments in the second paragraph of Section 4.5, which show that clipping the largest singular values alone recovers most, if not all, of IsoLoCo's benefits.

Furthermore, Appendix A.3 of the Iso-C paper [22] establishes a theoretical connection between the Subspace Alignment Ratio (SAR) of individual and merged matrices and representation-level interference after merging, and shows how singular-value equalization can improve this alignment. Although this result concerns model merging rather than distributed optimization, it provides additional theoretical motivation for applying Iso-C to aggregated pseudo-gradients, suggesting that spectral equalization may similarly mitigate interference between worker updates.

E Comparison and Compatibility with SparseLoCo

SparseLoCo [31] is a variant of DiLoCo which uses error feedback in combination with top-k sparsification and quantization to reach strong compression ratios with minimal performance degradation. We find that IsoLoCo is compatible with SparseLoCo but requires some modifications: since SparseLoCo does not use an outer momentum, the isotropic correction (SVD or Newton-Schulz) needs to be applied directly to the aggregated pseudo-gradients. At 178M model scale with $R = 8$ we find that IsoLoCo outperforms SparseLoCo and that SparseLoCo can actually benefit from the Iso-C correction. These results are presented in Table 11.

Table 11: Validation loss for DiLoCo, IsoLoCo, and SparseLoCo variants.

Method	**Valid loss**
DiLoCo	2.950
IsoLoCo (ours)	**2.889**
SparseLoCo	2.960
SparseLoCo + Iso-C (ours)	2.950

F IsoLoCo Has a Larger Critical Batch Size than DiLoCo

The critical batch size (CBS) characterizes the largest batch size that can be used without substantially degrading training performance [23, 25, 33]. Given sufficient parallel compute, a larger CBS enables more data to be processed in parallel, reducing the number of sequential optimization steps required for a fixed training-token budget and potentially decreasing wall-clock training time. Following [33], we define the CBS as the largest batch size achieving a final validation loss within 1% of the loss obtained at the optimal batch size.

We evaluate DiLoCo and IsoLoCo at the 178M model scale, varying the global batch size and tuning the inner learning rate, outer learning rate, and outer momentum for each configuration (as described in Section 3.1). As shown in Figure 5, IsoLoCo achieves a $2\times$ larger CBS than DiLoCo (1M versus 512K tokens). Remarkably, IsoLoCo at its CBS of 1M tokens still achieves a lower validation loss than DiLoCo at its optimal batch size of 256K tokens, despite using a $4\times$ larger batch size.

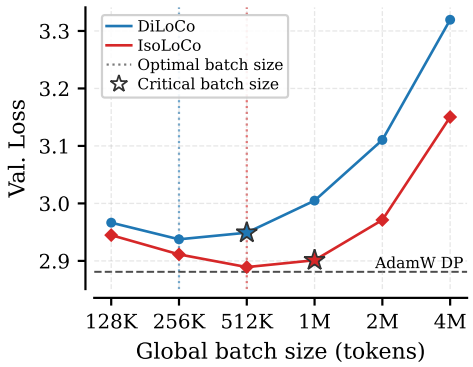

Figure 5: **IsoLoCo has a $2\times$ larger critical batch size than DiLoCo at the 178M model scale.** Final validation loss versus global batch size is shown, with the inner learning rate, outer learning rate, and outer momentum tuned independently for each method and batch size.

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: The main claims of the abstract and introduction directly reflect the presented results.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: The limitations of our work are explicitly discussed in Section 5.

   Guidelines:

   - The answer [N/A] 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: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] 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: The paper discloses all the information needed to reproduce the main experiments: most of the methodology and set-up are described in Section 3.1, the optimal hyper-parameters found are presented in Appendix A and additional details are presented throughout the text whenever necessary.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The data and models used are publicly available. We provide the code alongside instructions to faithfully reproduce the main results in the supplemental material as well as in a publicly available GitHub repository .

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

Answer: [Yes]

Justification: The experimental settings are described in the main paper and are detailed enough to understand the experiments and the results. Further details are provided with the code or in the appendix whenever deemed necessary.

Guidelines:

- The answer [N/A] 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: Error bars are not reported because it would be too computationally expensive given the scale of our experiments and our limited available computational resources. However, our main results are reported across multiple settings with varying model sizes, worker counts R and number of inner steps H. Our main claims are not based on any single reported number but rather on general trends and the consistent improvements provided by our method over the considered baselines across these various settings.

Guidelines:

- The answer [N/A] 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: See Section 3.1.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Our work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] 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: See Section 5.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work poses no such risks.

    Guidelines:

    - The answer [N/A] 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: The creators of all the assets used in this paper are properly credited, with the original papers cited. All assets used are open-source assets.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were only used for writing, editing and formatting purposes. The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.