OVERCOMING THE COMMUNICATION-PERFORMANCE TRADEOFF IN LLM PRETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Communication-efficient distributed training algorithms (e.g., DiLoCo) have received considerable interest due to their benefits for training large language models (LLMs) in bandwidth-constrained settings, such as across datacenters and over the internet. While these local-update methods achieve communication reduction through reduced synchronization, they still require communicating dense model-sized pseudo-gradients, resulting in a communication bottleneck even for cross-datacenter links. While quantization is often applied to reduce the pseudo-gradient's size, in the context of LLM pre-training, existing approaches have not been able to leverage sparsification without incurring significant performance degradation. In this work, we introduce *SparseLoCo*, a communication-efficient training algorithm for LLMs that *can effectively leverage* TOP-k *sparsification* and 2-bit quantization to reach extreme sparsity in the communicated pseudo-gradient, *as high as 97–99%*, while achieving lower final loss than dense DiLoCo. In our empirical study of language model pre-training, we demonstrate that SparseLoCo's improvements over DiLoCo in performance and pseudo-gradient compression are maintained across dense model scales (178M–2B), an MoE transformer (645M-A273M), increasing the number of workers, and increasing communication intervals.

1 INTRODUCTION

Frontier language models pre-trained on internet-scale data have led to considerable breakthroughs in recent years. However, due to their growing parameter counts, effectively training these models across expensive datacenter hardware while retaining efficiency—a central goal due to the resources spent on these runs—is becoming increasingly challenging. On the other hand, with the increasing availability of globally distributed computational infrastructure, the pre-training of large-scale models over the internet has recently garnered interest (Jaghouar et al., 2024a). Similar to training over the internet, pre-training across multiple datacenters requires mitigating the large communication overhead incurred by aggregating updates between workers.

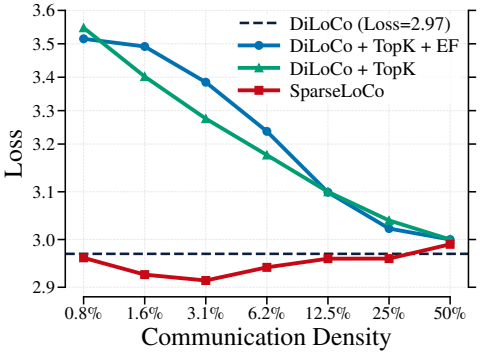
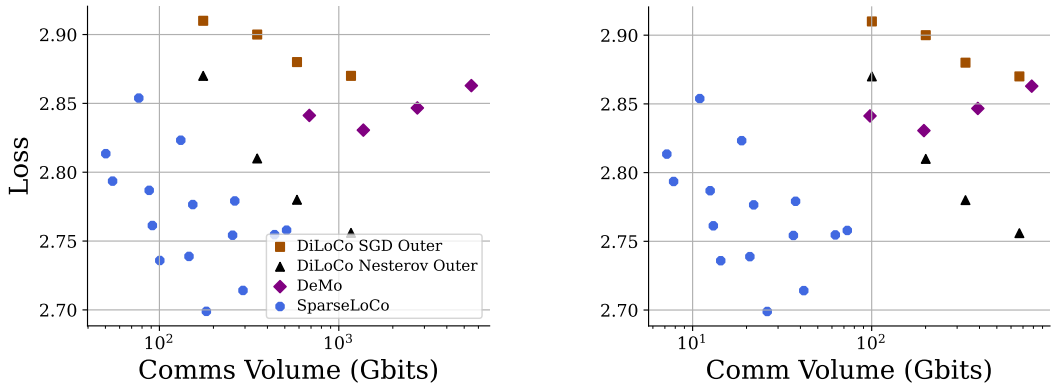


Figure 1: **Naive TOP-k compression hurts DiLoCo, even with error feedback (EF), whereas *SparseLoCo* outperforms DiLoCo with drastic communication reduction.** Final validation loss is shown versus communication density for a 178M LLaMA-2 model with R=8 workers and H=15 inner steps.

In LLM pre-training, several approaches have been proposed to reduce data-parallel communication cost. These include DiLoCo (Douillard et al., 2023), a variant of LocalSGD (Stich, 2018; Reddi et al., 2020), as well as methods that compress communicated tensors utilizing error feedback (Peng et al., 2024; Ahn et al., 2025; Wang et al., 2023) to mitigate information loss. These techniques have the complementary advantages of (1) reducing the communication frequency and (2) reducing the size of communicated messages. Combining the two is potentially advantageous for bandwidth-constrained settings like training over the internet or across datacenters. However, existing works focused on LLM pre-training with DiLoCo (Charles et al., 2025; Jaghouar et al., 2024b) do not take full advantage of compression schemes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Ring Communication (b) Parameter Server

Figure 2: **SparseLoCo lies on the Pareto frontier between loss and communication volume.** We report communication volume (outbound) for two settings: (A) ring communication topology (ring all-gather for SparseLoCo and DeMo, ring all-reduce for DiLoCo) and (B) parameter server. The points consider different H for DiLoCo, different densities for DeMo, and combinations of both for SparseLoCo using 512M models. We observe that, in both cases, SparseLoCo is at the Pareto frontier.

One of the most efficient compression schemes in the literature is TOP-k sparsification; however, in this work we observe that naively combining TOP-k with DiLoCo's outer momentum leads to large performance degradation, whereas error feedback naturally acts as a local approximation of outer momentum. Building on this, we introduce **SparseLoCo**, which replaces global outer momentum with a single error feedback accumulator, thereby unifying infrequent and sparsified communication. This enables aggressive TOP-k sparsification and quantization of pseudo-gradients, while outperforming full-communication DiLoCo and frequency-compressed baselines. Our contributions can be summarized as follows. In an LLM pre-training setting:

- We show an empirical connection between DiLoCo's outer momentum and error feedback with sparse compression.
- While existing gradient compression works *only* reduce communication cost, and the DiLoCo literature treats outer momentum as central to performance, we demonstrate that sparsifying the pseudo-gradient while treating the local momentum as an error feedback mechanism *substantially improves convergence* in this setting.
- We introduce SparseLoCo, a novel method that blends the benefits of multi-iteration methods like DiLoCo with TOP-k sparsification, quantization, and error feedback to achieve a Pareto-optimal tradeoff between performance and communication cost.
- Through extensive empirical evidence, we demonstrate that SparseLoCo significantly reduces communication volume compared to existing communication-efficient LLM training methods (e.g., DiLoCo and DeMo), while simultaneously outperforming them.

2 RELATED WORK

Federated Learning. In the federated learning (FL) literature, communication efficiency has been a central focus from the outset, as participants often operate over highly constrained and heterogeneous networks. A canonical example is Federated Averaging (FedAvg) (Konečnỳ et al., 2016; McMahan et al., 2017), which reduces communication frequency by performing multiple local updates before averaging model parameters. Related to our work, Mitchell et al. (2022) consider pseudo-gradient compression in the FL setting. However, while our work shares federated learning's emphasis on reducing communication overhead, it differs fundamentally in scope: we focus on large-scale pre-training of LLMs in settings with homogeneous data partitions (e.g., sharded web-scale corpora), where heterogeneity-mitigation strategies are unnecessary while achieving performance that can match standard data parallel schemes at equivalent FLOPs is paramount (Douillard et al., 2023).

LocalSGD and extensions to LLM training. Local Stochastic Gradient Descent (LocalSGD) (Stich, 2018) is a widely studied approach for reducing communication in distributed training by allowing

workers to perform multiple local updates before synchronizing. Stich (2018) formally introduced the method and proved its convergence, while Lin et al. (2018) highlighted that LocalSGD can lead to improved generalization compared to simply increasing the batch size. More recently, DiLoCo (Douillard et al., 2023; Charles et al., 2025) adapted the LocalSGD framework to large language model (LLM) pre-training, demonstrating that replacing the inner optimizer with AdamW and using a Nesterov momentum outer optimizer can yield substantial benefits. Our work builds upon this line of research by enabling aggressive TOP-k sparsification of the communicated pseudo-gradients in a LocalSGD-style framework, something that prior methods have not achieved while maintaining or improving upon state-of-the-art LLM training performance. Finally, Douillard et al. (2025) and Fournier et al. (2024) communicate a small subset of model parameters more frequently. SparseLoCo, on the other hand, maintains the benefit of infrequent communication while communicating a small-sized message and maintaining model synchronization; it can also be combined with approaches like Douillard et al. (2025).

Error Feedback and Compressed Updates. Error feedback (EF) has been extensively studied, particularly from a theoretical perspective, as a means to compensate for the information loss introduced by various gradient compression methods (Seide et al., 2014; Karimireddy et al., 2019; Stich & Karimireddy, 2019). EF has also been combined with various compression techniques, including quantization, sparsification (Shi et al., 2019), and low-rank approximation (Vogels et al., 2019; Ahn et al., 2025). Among LLM-oriented compressed optimizers, DeMo (Peng et al., 2024) considers a novel DCT encoding scheme alongside TOP-k compression combined with momentum SGD in the LLM setting, demonstrating it can achieve competitive performance, but without incorporating local updates or the ability to leverage adaptive optimizers. Closer to our multi-iteration setting, QSparseLocalSGD (Basu et al., 2019) is, to our knowledge, one of the few works that combine multi-iteration methods such as LocalSGD with error feedback, but is focused on theoretical analysis with non-adaptive optimizers and without outer momentum, which are crucial for high performance in the LLM setting. In contrast, our work targets the LLM pre-training regime and develops a method to combine aggressive TOP-k compression and error feedback as an efficient approximation of outer momentum which can improve performance while reaching extreme sparsity.

3 METHODOLOGY

In this section, we first review DiLoCo. We then introduce a per-replica local outer momentum (LOM) formulation and establish its equivalence to standard DiLoCo. This formulation allows us to investigate the connection between outer momentum and error feedback. Finally, we present **SparseLoCo**, which combines TOP-k sparsification and quantization with infrequent communication.

3.1 BACKGROUND AND NOTATION

Consider the DiLoCo/FedOpt (Douillard et al., 2023; Reddi et al., 2020) framework, which uses the following basic rule on each worker or replica to produce a pseudo-gradient at each outer step, $\Delta_r^{(t)}$:

$$\theta_r^{(t)} \leftarrow \mathrm{InnerOpt}_H\left(\theta^{(t-1)};\, \mathcal{D}_r\right), \quad \forall r \in [R],$$
$$\Delta_r^{(t)} \leftarrow \theta^{(t-1)} - \theta_r^{(t)}.$$

Here, H corresponds to the number of inner steps of the optimizer (typically AdamW), and R is the number of replicas. DiLoCo, which corresponds to an instantiation of FedOpt with AdamW as the inner optimizer and outer (server) momentum using Nesterov (Dozat, 2016), is given as follows:

$$\bar{\Delta}^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^{R} \Delta_r^{(t)}, \quad m^{(t)} \leftarrow \beta m^{(t-1)} + \bar{\Delta}^{(t)}, \quad \tilde{\Delta}^{(t)} \leftarrow \bar{\Delta}^{(t)} + \beta m^{(t)},$$
$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\tilde{\Delta}^{(t)}. \tag{1}$$

where $\tilde{\Delta}^{(t)}$ corresponds to the Nesterov direction; empirically, this has been shown to improve results (Douillard et al., 2023), and omitting it reduces the method to what is often called *LocalAdam* when Adam is used as the inner optimizer.

3.2 LOCAL OUTER MOMENTUM

We first propose a variant of DiLoCo that utilizes a per-replica local outer momentum instead of the unified global momentum. The goal of this algorithm is to provide insight into how well the

outer momentum can be locally approximated. We denote this algorithm DiLoCo-LOM (Local Outer Momentum):

$$m_r^{(t)} \leftarrow \beta m_r^{(t-1)} + \Delta_r^{(t)}, \quad \tilde{\Delta}_r^{(t)} \leftarrow \Delta_r^{(t)} + \beta m_r^{(t)}, \quad \tilde{\Delta}^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^{R} \tilde{\Delta}_r^{(t)},$$
$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \tilde{\Delta}^{(t)}.$$

Here, the outer momentum is updated locally, solely based on the local pseudo-gradient, while the final update is based on the average of the local Nesterov directions, $\tilde{\Delta}^{(t)}$. Note that typical implementations of DiLoCo store the outer momentum locally on each replica, meaning that DiLoCo-LOM does not add any memory overhead compared to the global momentum variant. We show that the DiLoCo-LOM update exactly matches the DiLoCo update in Appendix N.

Building up to SparseLoCo, we consider an additional method, denoted DiLoCo-LOM-Sub-k, where the local momenta have their largest components removed at the end of each outer step using $m_r^{(t)} \leftarrow m_r^{(t)} - \text{TOP-}k(m_r^{(t)})$. This allows us to study the impact of TOP-k subtraction used in error feedback, without sparsifying the pseudo-gradient.

Algorithm 1: SparseLoCo

Require: initial parameters $\{\theta_r^{(0)}\}$, inner steps H, outer steps T, outer LR α, error-feedback momentum β, workers R, per-worker data D_r, initial error buffers $\{e_r^{(0)}=0\}_{r=1}^{R}$.

for $t \leftarrow 1$ **to** T **do**
  for all Workers $r \in [R]$ **do**
    Local inner loops
    $\theta_r^{(t)} \leftarrow \theta_r^{(t-1)}$
    for $h \leftarrow 1$ **to** H **do**
      Sample $x \sim \mathcal{D}_r$
      $L \leftarrow f(x, \theta_r^{(t)})$
      $\theta_r^{(t)} \leftarrow \text{AdamW}(\theta_r^{(t)}, \nabla L)$
    end for
    $\Delta_r^{(t)} \leftarrow \theta_r^{(t-1)} - \theta_r^{(t)}$
    Compression + Error Feedback
    $\hat{\Delta}_r^{(t)} \leftarrow Q(\text{TOP-}k(\beta e_r^{(t-1)} + \Delta_r^{(t)}))$
    $e_r^{(t)} \leftarrow \beta\, e_r^{(t-1)} + \Delta_r^{(t)} - \hat{\Delta}_r^{(t)}$
    Aggregate + Outer Update
    $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{s=1}^{R} \hat{\Delta}_s^{(t)}$
    $\theta_r^{(t)} \leftarrow \theta_r^{(t-1)} - \alpha\, \Delta^{(t)}$
  end for
end for

3.3 SPARSELOCO

We now introduce SparseLoCo, which blends TOP-k sparsification and error feedback in place of the local outer momentum. We consider error feedback, e_r, applied to the pseudo-gradients:

$$e_r^{(t)} \leftarrow \beta e_r^{(t-1)} + \Delta_r^{(t)}, \quad \hat{\Delta}_r^{(t)} \leftarrow Q\Big(\text{TOP-}k\Big(e_r^{(t)}\Big)\Big),$$
$$e_r^{(t)} \leftarrow e_r^{(t)} - \hat{\Delta}_r^{(t)}, \quad \Delta^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^{R} \hat{\Delta}_r^{(t)}, \tag{2}$$
$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\, \Delta^{(t)}.$$

Here, Q is the quantization function, which allows further compression of selected values (implementation details are given in Section 4). When k is sufficiently small, the outer error feedback (OuterEF) closely approximates the local outer momentum in LOM, since only a few components will be subtracted from e_r. On the other hand, unlike LOM and LOM-Sub-k, SparseLoCo only aggregates quantized sparse vectors, drastically reducing the message size needed for communication. The full algorithm for SparseLoCo is given in Algorithm 1.

SparseLoCo uses a chunk-wise variant of the TOP-k operation inspired by Xu et al. (2021); Peng et al. (2024). To do so, we partition each 2D parameter tensor (e.g., attention and MLP weight matrices) into non-overlapping 64×64 blocks and each 1D tensor (e.g., layer-norm parameters) into contiguous chunks of size 4096, and then apply TOP-k independently within each chunk. This has three benefits compared to applying it at the full-tensor or global level: (a) the cost of naively storing indices for transmission is significantly reduced as each chunk's index space is bounded (further discussed in Appendix J). (b) TOP-k applied to entire models or individual tensors can overemphasize correlated variables; thus, chunking can have benefits for performance as further discussed in Appendix G. (c) Finally, chunking simplifies integration with tensor parallelism (TP) and Fully Sharded Data Parallel (FSDP) training, where parameters are partitioned across devices, by avoiding inefficient global TOP-k over entire parameters.

Table 1: SparseLoCo compared to DiLoCo and gradient compression (DeMo). We show the size of the pseudo-gradients sent, the number of synchronizations, quantization used, and loss. SparseLoCo outperforms other communication-efficient baselines in both communication efficiency and loss. All results are reported for 512M LLaMA-2 models pre-trained on a 10B-token budget with R=8 replicas and H=15 inner steps. Here, *Density* denotes the communication density as the percentage of coordinates in the pseudo-gradient vectors that are non-zero (and therefore transmitted) at each synchronization step.

Method	**Density**	**Loss**	**Pseudo-Grad Size**	**# of Syncs**	**Quantization**
AdamW DDP	100%	2.69	1.02 GB	2445	16-bit
DiLoCo (H=15)	100%	2.76	512.40 MB	163	8-bit
DeMo	0.78%	2.83	10.01 MB	2445	8-bit
DeMo	3.12%	2.86	40.03 MB	2445	8-bit
SparseLoCo (H=15)	0.78%	2.79	5.45 MB	163	2-bit
SparseLoCo (H=15)	3.12%	**2.70**	17.21 MB	163	2-bit

4 EXPERIMENTS

Our experiments use 178M-, 512M-, and 2B-parameter LLaMA-style decoder-only transformers on DCLM (Li et al., 2024) using the LLaMA-2 tokenizer (Touvron et al., 2023). Following Hoffmann et al. (2022), we allocate a token budget equal to $20\times$ the model size. Our experimental protocol follows Charles et al. (2025). Unless otherwise stated, our main results are reported on the 512M-parameter model with R=8 workers, per-worker batch size B=256, and sequence length L=2048, yielding a global batch of $B \times L \times R \approx 4.19$M tokens per step with communication interval H=15. For SparseLoCo, we employ 2-bit quantization with a chunk size of 4096 (non-overlapping square 64×64 grids for 2D parameters) and error feedback coefficient $\beta = 0.95$. After chunk-wise TOP-k, we quantize all selected values from the same tensor together. We follow the statistical quantization scheme used in Intellect-1 (Jaghouar et al., 2024a): values are centered by their mean, the range $[-6\,\sigma, 6\,\sigma]$ is divided into uniform bins (four bins for 2-bit quantization), and each bin stores the centroid of its assigned values as a lookup table. We dequantize the transmitted values before aggregation and the outer update. We apply a short error feedback (OuterEF) freeze, where the error feedback e_r is not utilized for the first 5% of the outer steps to improve training stability and performance (ablated in Table 11 in Appendix H). We also ablate EF decay β in Appendix I. Finally, we study scaling across model sizes (178M: Appendix K; 2B: Table 4) and the number of workers $R \in \{16, 32\}$ in Table 6 and Appendix Tables 13 and 14. We report the hyperparameter sweep ranges, architectural details, and selected configurations in Appendix R. As baselines, we include DiLoCo and DeMo—two strong communication-efficient methods (one using local iterations and the other using compression with error feedback) for LLMs—as well as a DDP AdamW baseline.

4.1 TOP-k COMPRESSION AND EF WITH LOCAL OPTIMIZERS AND LLMS

We start by demonstrating the unique properties of TOP-k compressors combined with error feedback in the setting of LLMs and local optimizers. We consider DiLoCo with an SGD outer optimizer or, as it is referred to in the literature, LocalAdam. Similar to Basu et al. (2019), which uses CNNs and SGD local optimizers, we apply a number of compressors with full-memory error feedback (β=1). This corresponds to using different compressors in (2). The results are shown in Figure 3. As confirmed in prior literature, we observe that combining error feedback and compression can lead to reduced communication cost while maintaining performance or incurring modest convergence degradation. Specifically, with aggressive quantization (2 bits), performance does not degrade, while Random-k sparsification with 6.25% density leads to a slight degradation. On the other hand, we observe that a unique phenomenon arises when TOP-k sparsification is combined with error feedback: both final performance and convergence speed improve substantially.

We emphasize that this is, to the best of our knowledge, a unique property of the combination of TOP-k and EF that has not been observed in the literature on compression and error feedback.

We hypothesize that this improvement is due to the error feedback buffer providing benefits similar to the outer momentum in DiLoCo. Motivated by this result, we further investigate this connection,

and then evaluate SparseLoCo's performance against strong baselines across multiple settings and provide a thorough ablation study.

4.2 Building intuition with Local Outer Momentum

We use the DiLoCo-LOM and DiLoCo-LOM-Sub-k algorithms to empirically link the standard outer momentum in DiLoCo to the error feedback mechanism in SparseLoCo. In DiLoCo-LOM, each replica maintains a local outer momentum accumulator that is averaged only at synchronization. In DiLoCo-LOM-Sub-k, we subtract the largest-magnitude entries of the local momentum after each synchronization. Here, the local momentum closely mirrors the error feedback in Top-k compression, as both are maintained locally, and in both cases the largest-magnitude Top-k entries are repeatedly removed, while DiLoCo-LOM-Sub-k keeps the communicated pseudo-gradients dense. As shown in Table 2, DiLoCo-LOM matches DiLoCo's loss—consistent with Proposition 1 (Appendix N)—and pruning 25% of the largest accumulator entries has a negligible impact, whereas removing outer momentum entirely degrades performance. This experiment suggests momentum benefits are maintained even with aggressive pruning of the largest values in the outer momentum or error feedback.

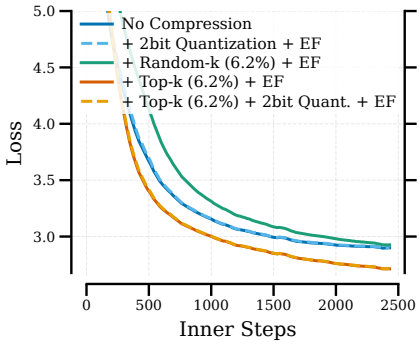


Figure 3: **Top-k sparsification with error feedback improves LocalAdam in LLM pre-training.** We plot validation loss for LocalAdam (DiLoCo with SGD outer optimizer) under different pseudo-gradient compressors combined with error feedback. Quantization does not impact performance (curves closely overlap full-precision counterparts), Random-k slightly degrades, while Top-k substantially improves both convergence speed and final loss. All results report 512M models pre-trained on a 10B-token budget with R=8 replicas and H=15 inner steps.

To quantify how closely local outer momentum tracks the target global outer momentum in DiLoCo, we maintain a reference global accumulator and, over the first 20 outer steps, compute the cosine similarity between this reference and each replica's local accumulator at corresponding steps. The average similarity between individual local accumulators and the global reference accumulator is ≥ 0.75 for DiLoCo-LOM-Sub-k (25%), indicating that removing the top components from local accumulators each outer step still allows them to remain a strong directional proxy for the global momentum, supporting our interpretation of SparseLoCo's error feedback state as a local approximation of DiLoCo's outer momentum.

4.3 Naive Top-k for DiLoCo

We now present two straightforward ways to incorporate Top-k sparsification into DiLoCo and explain why they do **not** work. The first (DiLoCo + Top-k) simply applies the Top-k operation before communication. The second (DiLoCo + Top-k + EF) adds error feedback to track any non-communicated information. These modifications correspond to applying Top-k and optional error feedback to the local pseudo-gradients immediately before Eq. 1. We summarize the DiLoCo variants and provide pseudocode for further clarification in Appendix P.

Table 2: **Sparse local outer momentum (DiLoCo-LOM-Sub-k) matches DiLoCo's global outer momentum in final loss.**

Method	Loss
DiLoCo	2.760
DiLoCo w.o. outer momentum	2.868
DiLoCo-LOM	2.759
DiLoCo-LOM-Sub-k (25%)	2.761

In Figure 1, we plot final loss as a function of communication density for SparseLoCo and these naive Top-k DiLoCo variants. We find that SparseLoCo using only a local EF can outperform all methods at low communication density, including dense DiLoCo. In contrast, the performance of DiLoCo + Top-k and DiLoCo + Top-k + EF substantially degrades at high sparsity. We hypothesize that this is caused by the outer momentum preserving directions corresponding to the large-magnitude components, which then dominate subsequent updates.

4.4 SparseLoCo Evaluations

Table 1 compares SparseLoCo at H=15 against existing state-of-the-art methods for pre-training LLMs. We use 2-bit quantization for SparseLoCo and observe no loss degradation, while using the prescribed quantization settings for baselines (Douillard et al., 2023; Peng et al., 2024). We observe

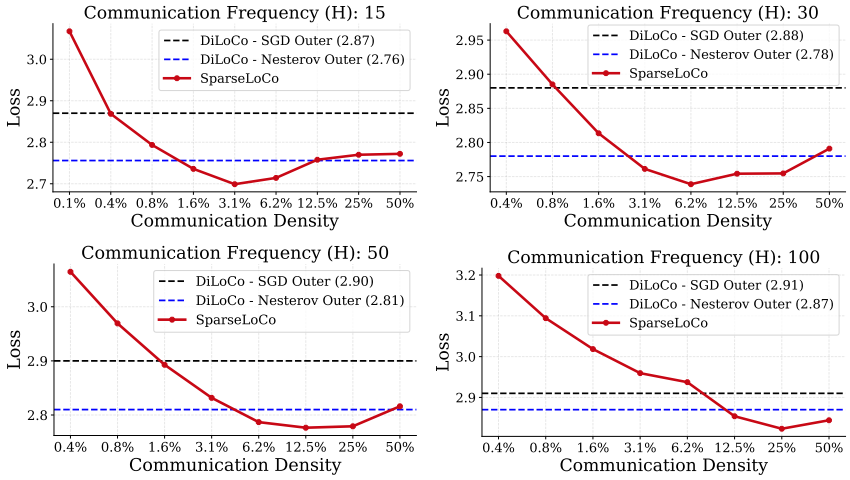


Figure 4: **SparseLoCo outperforms DiLoCo for $H \in \{15, 30, 50, 100\}$ communication intervals.** We evaluate SparseLoCo, DiLoCo, and DiLoCo with SGD outer optimizer across different communication intervals, and different sparsity levels for SparseLoCo. We report the best performance in each case. Crucially, SparseLoCo can outperform DiLoCo while communicating significantly less. We also observe that the optimal density grows with higher communication intervals. All experiments were conducted using 512M models.

Table 3: Benchmark (0-shot) accuracy (higher is better). **Best** is bold. We evaluate the same 512M pretrained models as in Table 1, using the best-performing models for SparseLoCo and DeMo. We observe that SparseLoCo outperforms the DeMo and DiLoCo baselines across all benchmarks.

Method	ARC-Easy	HellaSwag	PIQA
AdamW DDP	44.99%	36.08%	65.34%
DiLoCo	44.28%	34.50%	64.96%
DeMo	41.92%	32.37%	64.09%
SparseLoCo	**45.24%**	**36.49%**	**65.23%**

that SparseLoCo obtains lower final loss than the DiLoCo and DeMo baselines, while enjoying the simultaneous communication benefits of aggressively sparsified pseudo-gradients and reduced synchronization frequency. As SparseLoCo inherently utilizes error feedback, it further reduces communication size by quantizing the sparsified values. We further compare the performance on simple downstream tasks relevant at this model scale in Table 3, demonstrating that the performance improvements are consistent.

SparseLoCo performance at different sparsity levels and communication intervals. In Figure 4, we further demonstrate the performance improvements of SparseLoCo across TOP-k densities and increasing $H \in \{15, 30, 50, 100\}$ values compared to well-tuned DiLoCo and DiLoCo without outer momentum baselines. We observe a trend that aligns with the hypothesis that SparseLoCo's OuterEF can provide similar benefits to DiLoCo's outer momentum. In particular, we first observe that not using outer momentum in DiLoCo leads to significant performance degradation and that this setting corresponds exactly to the fully dense case for TOP-k (i.e., k=100%). With SparseLoCo, we observe that (i) extreme sparsity levels (when nearly nothing is sent, e.g., 0.05%) degrade performance. (ii) With increasing density (while remaining sparse), performance improves and eventually exceeds that of DiLoCo for all values of H. At these density levels, the EF buffer remains relatively dense and accumulates residual gradients, resembling a sparsified outer momentum in accumulating gradients. (iii) Finally, as k approaches dense communication, the EF buffer becomes more sparse (due to line 11 in Algorithm 1), trending towards the performance of DiLoCo with no outer momentum and thus degrading performance. The same three regimes appear across all communication intervals H. Furthermore, in Figure 7 in Appendix O, we observe the same phenomenon with DeMo (Peng et al., 2024) as well, while the overall performance is inferior to SparseLoCo.

Table 4: Evaluation loss and benchmark (0-shot) accuracy of 2B-parameter LLMs with R=16 contributing peers. Best in **bold**.

Method	Loss	ARC-Easy	ARC-Challenge	HellaSwag	PIQA	WinoGrande
AdamW DDP	2.34	58.42%	32.00%	56.87%	72.25%	56.59%
DiLoCo	2.37	**60.48%**	30.55%	54.95%	**72.85%**	55.56%
SparseLoCo	**2.36**	59.05%	**32.17%**	**55.49%**	**72.85%**	**58.56%**

Table 5: **Ablation Studies.** (*Left*) SparseLoCo with Random-k vs. TOP-k sparsification: TOP-k significantly outperforms Random-k across all densities. (*Right*) Effect of quantization on loss: 2-bit shows almost no degradation vs. full precision.

Density	Rand-k Loss	TOP-k Loss
1.56%	3.05	2.74
3.12%	2.98	2.70
6.25%	2.93	2.71

Quant. bits	1	2	3	4	32
Loss	4.79	2.70	2.70	2.70	2.70

SparseLoCo can outperform DiLoCo. Across all communication intervals H, we observe regimes where SparseLoCo outperforms DiLoCo. A plausible explanation is that sparse aggregation at a well-chosen k emphasizes high-saliency components and reduces interference among updates, echoing intuitions from recent model-merging work in multi-task fine-tuning (Yadav et al., 2023; Davari & Belilovsky, 2024).

Higher sparsity is needed with fewer inner steps. In Figure 4, we observe a systematic pattern that optimal performance of SparseLoCo is reached at a higher sparsity level with fewer inner steps. This is consistent with the fact that more inner steps H communicate information from a larger total number of samples. Indeed, we would expect a trajectory with more steps to have a larger support.

SparseLoCo is at the Pareto frontier in communication volume. In Figure 2 we compare the communication volume of SparseLoCo to that of DiLoCo, DiLoCo w.o. outer momentum, and DeMo. The exact communication setting and the underlying implementation of the aggregation can have a significant impact on the communication volume. We consider two common setups from the literature—methods utilize either ring all-reduce or ring all-gather (Fig. 2 A), or a parameter server (Fig. 2 B). We observe that in both cases, SparseLoCo lies on the Pareto frontier while other methods have a strictly worse tradeoff. We note that the results in Fig. 2 A assume a naive all-gather implementation. Further communication savings may be possible, for example, by summing overlapping indices along steps in the all-gather ring or utilizing a specially designed all-reduce (Li & Hoefler, 2022).

SparseLoCo can be used with Ring All-Reduce as a drop-in for DiLoCo. Although our analysis and motivation in this work focus on aggressively compressing the per-iteration message size, we note that in communication settings where efficient all-reduce is already available and preferred, SparseLoCo still provides significant benefit over DiLoCo while incurring no additional memory overhead and negligible compute overhead. Concretely, the aggregation step in Algorithm 1, Line 12, can be performed directly by an all-reduce over a sparse vector. This has two significant benefits over DiLoCo with all-reduce (AR): (1) As observed in Table 16 and Figure 4, the performance is improved when k is optimally selected and (2) the outer error feedback naturally supports more aggressive quantization than naive DiLoCo, allowing for 2-bit quantization to be used *without an additional accumulator*, unlike Thérien et al. (2025).

SparseLoCo scales across model sizes, communication intervals, and the number of replicas. In Table 6, we evaluate scaling of DiLoCo and SparseLoCo with the number of workers $R \in \{8, 16, 32\}$ using a 512M-parameter model. Scaling beyond 8 replicas without significant degradation is a known challenge (Charles et al., 2025). We observe that SparseLoCo consistently outperforms DiLoCo with a larger number of workers across all settings and across a number of densities, showing that it can help address the challenge of scaling the number of replicas. We also observe that with more workers, a lower density can sometimes be supported. In Appendix K, we study the impact of the number of replicas at the 178M model size. We also evaluate SparseLoCo at a larger communication interval (H=250), using an overtraining regime with a doubled token budget following Charles et al. (2025),

Table 6: Final evaluation loss for a 512M model with communication interval $H{=}50$, varying the number of replicas $R \in \{8, 16, 32\}$ and SparseLoCo density. SparseLoCo consistently outperforms DiLoCo as R increases. AdamW DDP reaches 2.69 across all R. Best result per row is in **bold**.

	DiLoCo	**SparseLoCo Density**					
R	100%	0.78%	1.56%	3.12%	6.25%	12.50%	25.00%
8	2.81	2.97	2.89	2.83	2.79	**2.78**	**2.78**
16	2.87	3.00	2.92	2.86	2.82	**2.80**	2.84
32	2.93	3.09	3.00	2.92	**2.88**	2.91	3.02

where we again see that SparseLoCo is able to achieve competitive performance with DiLoCo while reducing communication. Finally, we run a larger-scale 2B-parameter model using $R{=}16$ workers and communication interval $H{=}50$, where SparseLoCo with 6.25% density outperforms DiLoCo (Table 4). We observe that the benefits of SparseLoCo are maintained at this scale.

Mixture-of-Experts models. SparseLoCo can also be applied to MoE transformers without any algorithmic changes. On a Qwen2-MoE-style 645M-A273M model, we observe that SparseLoCo at 3.12% density (loss=2.741) closes the gap to AdamW DDP training (loss=2.739) while substantially outperforming DiLoCo (loss=2.803). See Appendix C for additional details and results.

Wall-clock time and compute utilization. Figures 5 and 6 in Appendices E and F provide training wall-clock time and compute utilization results, showing strong communication efficiency of SparseLoCo.

Large-scale real-world deployment. SparseLoCo was used in a real-world, collaborative, globally distributed training run of a 72B-parameter LLM in a low-bandwidth environment, representing the largest deployment of this kind reported to date—roughly $7\times$ the scale of Intellect-1 (Jaghouar et al., 2024a). The downstream performance of the resulting model is competitive with similar models trained and open-sourced in high-bandwidth cluster settings. Here, the use of all-gather with extremely small payloads allows fault-tolerant implementations that are easily robust to peer churn and node failures; completed uploads remain accessible after sender failures, and missing or invalid contributions can be excluded without rebuilding a ring or retrying collectives as in ring all-reduce. Additional deployment details and empirical measurements are provided in Appendix A.

4.5 Ablations

We now highlight key design choices of SparseLoCo through a series of ablations.

Random-k. We ablate the choice of Top-k compared to the alternative Random-k in Table 5. We observe that performance is significantly degraded when using Random-k for the same number of indices selected, consistent with prior observations (Shi et al., 2019; Wang et al., 2023), emphasizing the importance of this design choice.

Quantization. As discussed, SparseLoCo benefits from stronger quantization than non-EF methods and supports quantization down to 2 bits and was generally observed to give results very close to full precision. In Table 5, we show the performance at different quantization values, showing that 2-bit quantization can be achieved at almost no performance cost.

5 Conclusion

We have proposed SparseLoCo, an algorithm that combines infrequent communication with aggressive Top-k sparsification and quantization of pseudo-gradients for LLM pre-training. Our work establishes that DiLoCo's outer momentum can be reformulated using local momentum accumulators. By connecting local momentum with error feedback under aggressive sparsification, we develop *SparseLoCo*. Our extensive experiments confirm that SparseLoCo significantly reduces communication while outperforming strong baselines such as DiLoCo and DeMo, placing it on the Pareto frontier of loss versus communication volume. Additionally, our experiments reveal that sparse aggregation may actually be useful for improving performance, opening the possibility of studying more sophisticated aggregation methods in the pre-training setting.

AI USE STATEMENT

We used large language models solely for language editing (grammar, vocabulary, and phrasing). All suggested edits were carefully reviewed by the authors before incorporating to the main text.

ETHICS STATEMENT

The paper conforms to the ICLR Code of Ethics. SparseLoCo is a communication-efficient method that enables low-bandwidth pretraining over geographically distributed compute resources, and thus does not impose any negative societal impact on its own.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we share an anonymized code repository containing the implementation, training scripts, and configuration files needed to reproduce the main experiments in the supplementary materials. Our main experiments use the public DCLM (Li et al., 2024) dataset for which we provide data processing scripts. Furthermore, we provide the full SparseLoCo algorithm, quantization/compression details, training setup, model architectures, token budgets, and all other relevant hyperparameters and search spaces in Sections 3–4 and Appendices C, D, M, R, and other related experimental appendices.

REFERENCES

Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates. *arXiv preprint arXiv:2504.05295*, 2025.

Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems*, 32, 2019.

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

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

MohammadReza Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In *European Conference on Computer Vision*, pp. 270–287. Springer, 2024.

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

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

Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *Proceedings of the 4th International Conference on Learning Representations, Workshop Track*, 2016. URL https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ.

Ilyas Fatkhullin, Alexander Tyurin, and Peter Richtárik. Momentum provably improves error feedback! *Advances in Neural Information Processing Systems*, 36:76444–76495, 2023.

Louis Fournier, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky, and Edouard Oyallon. Wash: Train your ensemble with communication-efficient weight shuffling, then average. *arXiv preprint arXiv:2405.17517*, 2024.

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.

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

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

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

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

Charles-Étienne Joseph, Benjamin Thérien, Abhinav Moudgil, Boris Knyazev, and Eugene Belilovsky. Meta-learning optimizers for communication-efficient learning. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=uRbf9ANAns.

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

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

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pp. 5132–5143. PMLR, 2020.

Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. *arXiv preprint arXiv:2304.13960*, 2023.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Shigang Li and Torsten Hoefler. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 135–149, 2022.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (eds.), *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020.

Joel Lidin, Amir Sarfi, Evangelos Pappas, Samuel Dare, Eugene Belilovsky, and Jacob Steeves. Incentivizing permissionless distributed learning of llms. *arXiv preprint arXiv:2505.21684*, 2025.

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

Zhengzhong Liu, Bowen Tan, Hongyi Wang, Willie Neiswanger, Tianhua Tao, Haonan Li, Fajri Koto, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. Llm360 k2: Building a 65b 360-open-source large language model from scratch. *arXiv preprint arXiv:2501.07124*, 2025.

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*, pp. 1273–1282. PMLR, 2017.

Nicole Mitchell, Johannes Ballé, Zachary Charles, and Jakub Konečnỳ. Optimizing the communication-accuracy trade-off in federated learning with rate-distortion theory. *arXiv preprint arXiv:2201.02664*, 2022.

Jose Javier Gonzalez Ortiz, Jonathan Frankle, Mike Rabbat, Ari Morcos, and Nicolas Ballas. Trade-offs of local sgd at scale: An empirical study. *arXiv preprint arXiv:2110.08133*, 2021.

Bowen Peng, Lizhang Chen, Baiyu Su, Jeffrey Quesnelle, Diederik P Kingma, and Qiang Liu. Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International conference on artificial intelligence and statistics*, pp. 2021–2031. PMLR, 2020.

Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pp. 8253–8265. PMLR, 2020.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In Haizhou Li, Helen M. Meng, Bin Ma, Engsiong Chng, and Lei Xie (eds.), *15th Annual Conference of the International Speech Communication Association, INTERSPEECH 2014, Singapore, September 14-18, 2014*, pp. 1058–1062. ISCA, 2014. URL https://doi.org/10.21437/Interspeech.2014-274.

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

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

Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350*, 2019.

Benjamin Thérien, Xiaolong Huang, Irina Rish, and Eugene Belilovsky. Muloco: Muon is a practical inner optimizer for diloco, 2025. URL https://arxiv.org/abs/2505.23725.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 14236–14245, 2019.

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

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

Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. Grace: A compressed communication framework for distributed machine learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pp. 561–572, 2021. doi: 10.1109/ICDCS51616.2021.00060.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 2023. URL https://arxiv.org/abs/2306.01708.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Hanzhen Zhao, Xingyu Xie, Cong Fang, and Zhouchen Lin. Separate: A simple low-rank projection for gradient compression in modern large-scale model training process. In *The Thirteenth International Conference on Learning Representations*, 2025.

A Real-World Deployment for Collaborative Permissionless Distributed Training Over the Internet

SparseLoCo has been deployed in a real-world setting and is being used to collaboratively train models up to 8B and 72B with permissionless global participants using an incentive scheme (Lidin et al., 2025) that rewards participants purely based on analysis of their compressed pseudo-gradients. This was done on top of an existing blockchain. The addition and coordination of peers and their rewards were handled through the blockchain. Communication of pseudo-gradients was routed through globally distributed, S3-compliant object storage—specifically Cloudflare R2—which enabled rapid dissemination of model updates worldwide. This setup allowed updates to be time-stamped and verified as part of the reward mechanism (Lidin et al., 2025). Each peer maintained its own storage bucket, posting read credentials to the blockchain so that both other peers and the reward mechanism could access their compressed pseudo-gradients. This setup also provides strong fault tolerance and robustness to peer churn: missing uploads are skipped so the remaining peers can continue without rebuilding a ring or restarting the exchange, while completed uploads remain accessible even if their sender fails mid-communication.

SparseLoCo is particularly advantageous in this communication setup, as cloud providers have high bandwidth for peer downloads and are able to rapidly distribute and mirror files across the globe. On upload, each peer sends only its pseudo-gradient through the cloud provider. Therefore, its outbound communication (and required upload bandwidth) is kept low. Each peer then downloads the pseudo-gradients from the cloud provider, which is able to easily handle the high bandwidth constraints. An example of a practical communication time measured with an 8B model is 12 seconds on average, including sending its compressed pseudo-gradient and downloading other workers' messages, with the test node never exceeding 500 Mb/s. Compared to processing with $8\times$ H200, which takes around 4.5 minutes, leading to minimal wall-clock time degradation despite traffic over the internet. For reference, Jaghouar et al. (2024a), which trained a similarly sized model (10B) with 8-bit DiLoCo, reports an average globally distributed all-reduce synchronization time of approximately 10 minutes for a peak of $R{=}14$ nodes participating and a processing time of 38 minutes. We also performed test measurements of communication time for a 70B model with the same setup as above ($R{=}20$ peers), measuring a total communication time of 70 seconds on average, with the test node never reaching more than 500 Mb/s downlink and 110 Mb/s uplink.

72B LLM training. We have deployed SparseLoCo on the live system discussed above to train a 72B model, the largest collaborative foundation model training run ever considered (roughly $7\times$ the scale of the prior largest decentralized training run (Jaghouar et al., 2024a) with lower communication overhead as discussed above). The deployment uses a maximum of $R{=}20$ replicas, each associated with a peer. In such collaborative systems, the number of replicas constantly changes as peers join and leave at will or are excluded due to malicious behavior or incompetence, all of which may happen during a collective. These instabilities can make conventional ring all-reduce impractical, as changes in participation require communicator reconfiguration and may force ongoing collectives to restart. We instead opt for an all-gather implementation through object storage, achieving remarkable communication speed with low-bandwidth nodes as reported above. This enables tolerance to peer churn, faulty nodes, and bad-faith actors, as peer contributions can be validated before aggregation. We used $H{=}30$ inner steps and a global batch size of approximately 8M tokens per inner step. We utilized a peak learning rate of 1.2e-4 and the DCLM dataset. Although the system design allows peers to use any target hardware that can achieve reasonable throughput, the suggested hardware requirements targeted $8\times$ B200 GPUs per replica. Table 7 compares our Internet-trained 72B model against both decentralized (Intellect-1 (Jaghouar et al., 2024a)) and centralized (LLaMA-2 (Touvron et al., 2023) and LLM360 K2 (Liu et al., 2025)) large-scale baselines across standard zero-shot benchmarks. We observe that the SparseLoCo-trained model is able to achieve competitive performance, particularly given its token budget.

B Extended Related Work

Recent works study compression with error feedback for LLM training without local updates (Wang et al., 2023; Peng et al., 2024; Zhao et al., 2025). EF21-SGDM (Fatkhullin et al., 2023) analyzed how to combine error feedback with momentum, introducing a momentum-compatible variant that requires two accumulators and is largely focused on theoretical aspects and does not address the

Table 7: **Benchmark comparison across Internet-trained and centralized-cluster-trained LLMs.** We report zero-shot accuracy on ARC-C, ARC-E, HellaSwag, and MMLU. Internet-trained models are trained in a low-bandwidth setting, while centralized-cluster models are trained using conventional large-scale datacenter infrastructure and AdamW.

Model	Compute Env.	Size	Tokens	ARC-C	ARC-E	HellaSwag	MMLU
Intellect-1	Internet	10B	1T	44.8	71.6	70.5	32.7
Ours	Internet	72B	1.1T	56.8	80.9	80.6	67.1
LLM360 K2	Datacenter	65B	1.4T	53.84	75.93	82.81	63.90
LLaMA-2-7B	Datacenter	7B	2T	45.90	74.58	75.92	40.86
LLaMA-2-70B	Datacenter	70B	2T	57.59	80.77	83.86	65.56

multi-iteration setting or the practical challenges of LLM pre-training. Similarly, CocktailSGD (Wang et al., 2023) uses error feedback with multiple compression operators in an LLM fine-tuning setting, yet does not explore the integration of local iteration methods.

In federated learning, Rothchild et al. (2020); Reisizadeh et al. (2020) explore compressed updates through sketching or quantization. Beyond reducing communication overhead, numerous approaches such as SCAFFOLD (Karimireddy et al., 2020) and FedProx (Li et al., 2020) address the unique challenge of data heterogeneity—where each client's dataset may follow a different distribution—by introducing control variates or proximal terms to stabilize convergence.

Beyond basic LocalSGD, SlowMo (Wang et al., 2019) adds momentum to the SGD outer optimizer, while meta-learning approaches (Joseph et al., 2025) seek to learn an improved outer optimizer. However, these approaches were not shown to scale well to pre-training in Ortiz et al. (2021).

Other recent works focus on studying DiLoCo variants that also communicate optimizer states, allowing them to prove convergence and improve performance at the cost of additional communication (Cheng & Glasgow, 2025; Iacob et al., 2026b;a; Jovanović et al., 2026). Given our focus on extreme communication efficiency, in this work, we build upon the DiLoCo line of research and choose not to communicate optimizer states.

C Mixture-of-Experts Experiments

While our main experiments use dense models, SparseLoCo can be applied to MoE transformers without algorithmic changes. In this section, we evaluate SparseLoCo against baselines using a Qwen2-MoE-style architecture (Yang et al., 2024). The model has 645M total parameters and 273M active parameters per token (645M-A273M). The model uses 12 decoder layers, hidden size 1024, 8 attention heads, 8 key-value heads, and sequence length 2048, with an MoE FFN in every decoder layer. We use 32 routed experts with top-4 routing and routed-expert intermediate size 358 (granularity $1/8$), and one shared expert with intermediate size 2816. The router auxiliary loss coefficient is 0.001.

We sweep AdamW over learning rates $\{10^{-3}, \mathbf{3{\times}10^{-3}}, 5{\times}10^{-3}\}$ and $\beta_2 \in \{0.9, \mathbf{0.95}, 0.99\}$, and DiLoCo over $\alpha_{\text{inner}} \in \{10^{-3}, \mathbf{8{\times}10^{-4}}, 6{\times}10^{-4}\}$ and $\alpha_{\text{outer}} \in \{\mathbf{0.6}, 0.8, 1.0\}$. We did not run an MoE-specific sweep for SparseLoCo and instead reused the dense-model hyperparameters from Appendix R. As shown in Table 8, SparseLoCo nearly matches AdamW at 3.12% density and clearly outperforms DiLoCo.

Table 8: **MoE validation loss comparison.** We report final validation loss for the 645M-A273M Qwen2-MoE-style model. AdamW and DiLoCo use the best settings from their MoE-specific sweeps, while SparseLoCo reuses the dense-model hyperparameters from Appendix R. SparseLoCo nearly matches AdamW at 3.12% density and improves substantially over DiLoCo. Best result is **bolded**.

Method	Density	Loss
AdamW	100%	**2.739**
SparseLoCo	3.12%	2.741
SparseLoCo	1.56%	2.776
DiLoCo	100%	2.803

D STREAMING SPARSELOCO

In this section, we verify that Streaming DiLoCo (Douillard et al., 2025) and SparseLoCo can be combined. Streaming DiLoCo is an orthogonal direction to SparseLoCo for reducing peak communication volume by hiding it. SparseLoCo reduces the absolute number of bits communicated per step through compression, thus indirectly reducing peak communication. Streaming DiLoCo directly reduces peak bandwidth by only communicating subsets of the model's parameters at a time but does not reduce the absolute number of bits communicated.

Table 9 reports results for combining SparseLoCo with Streaming DiLoCo to reduce peak communication volume. We train 18-layer 1B-parameter transformers (hidden dimension 2048) in this ablation. The model is partitioned into three even subsets of 6 hidden layers, with the first and third subsets containing the embedding and unembedding layers, respectively. We train the $1,055$M-parameter model for a Chinchilla-optimal 21B tokens (Hoffmann et al., 2022). We use a communication interval of $H = 15$ for the full model (Streaming communicates every 5 steps) and 8 workers. We observe that both models reach the same final validation loss (they differ only in the 4th decimal), while Streaming SparseLoCo reduces peak communication volume by a factor of 3.

Table 9: We combine SparseLoCo with Streaming DiLoCo to reduce peak communication volume when training an 18-layer 1B parameter transformer. The model is partitioned into three even subsets of 6 hidden layers, with the first and third subsets containing the embedding and unembedding layers, respectively. We use a communication interval of $H = 15$ for the full model (Streaming communicates every 5 steps) and 8 workers. We observe that both models reach the same final validation loss (they differ only in the 4th decimal), while Streaming SparseLoCo reduces peak communication volume by a factor of 3.

Method	**Density**	**Comm. Volume/Step**	**Peak Comm. Volume**	**Loss**
SparseLoCo	3.125%	35.03 MB	35.03 MB	**2.51**
Streaming SparseLoCo	3.125%	35.03 MB	11.68 MB	**2.51**

E WALL-CLOCK TIME UNDER BANDWIDTH CONSTRAINTS

We measure wall-clock training time in our main 512M setting (R=8 workers, H=15, 10B-token budget, using $8\times$H100 GPUs) under a realistic $\sim$300 Mbps cross-replica bandwidth constraint (a common setting for machines rented from vast.ai). Figure 5 plots validation loss against wall-clock time. SparseLoCo reaches DiLoCo's final loss about two hours earlier. We also observed that the overhead of SparseLoCo's compression (TOP-k, quantization, chunking) is negligible: only 0.03% extra compute per outer step. Finally, as the pseudo-gradient size scales with model size, we expect the wall-clock gap to increase with larger models at the same bandwidth.

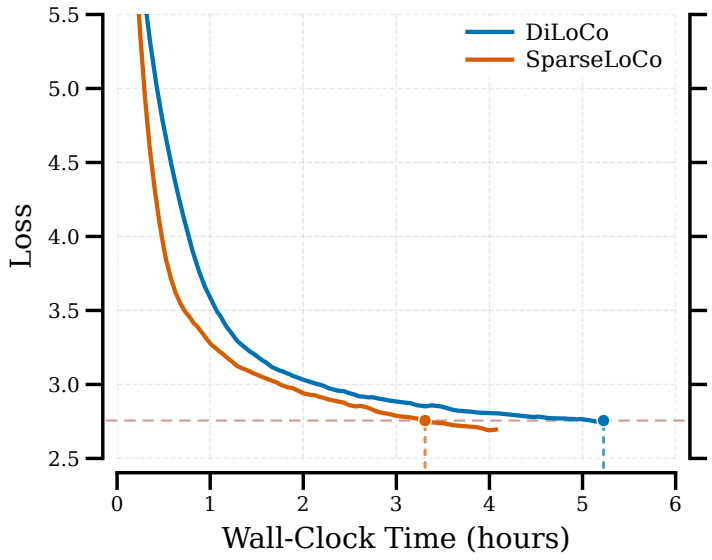


Figure 5: **Wall-clock improvement under a 300 Mbps bandwidth limit.** Validation loss vs. wall-clock time for a 512M model trained with H=15, R=8 workers, and a 10B-token budget. SparseLoCo (3.12% density) reaches DiLoCo's final loss roughly two hours earlier.

F Compute Utilization vs. Bandwidth

We estimate the compute utilization of each method as $\frac{T_{\text{compute}}}{T_{\text{compute}}+T_{\text{comms}}}$, where T_{compute} is the time spent in pure computation and T_{comms} is the time spent in communication. We first estimate T_{compute} considering the FLOPs profile of the model, assuming 8×B200 GPUs per worker with R=16 workers, a theoretical FP16/BF16 throughput of 2.25×10^{15} FLOPs/s per GPU, and a reasonable model FLOP utilization (MFU) of 40%. Then, we simulate training, calculating T_{comms} under different bandwidth constraints considering the pseudo-gradient message sizes of each method (Figure 6). At low bandwidths, SparseLoCo achieves substantially higher utilization than DDP, DeMo, and DiLoCo.

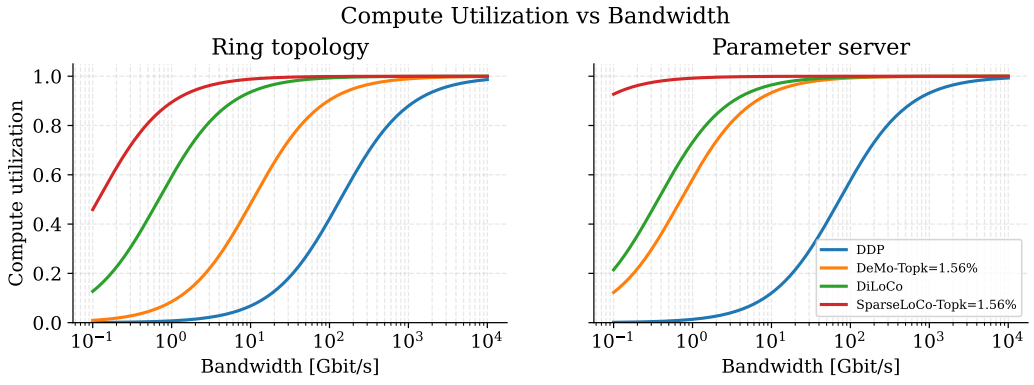


Figure 6: Compute utilization vs. bandwidth for DDP, DeMo, DiLoCo, and SparseLoCo under ring (left) and parameter-server (right) topologies. Compute utilization is calculated as total time spent in computation over full training time (including communication). We simulate a 70B LLaMA-2 model trained with R=16 replicas each with 8× B200, assuming a reasonable 40% MFU under different bandwidth settings.

G Effect of Chunking, DCT, and inner steps

A recently introduced method (Peng et al., 2024) considered the single-step setting with error feedback, using a compression function that first applies a discrete cosine transform (DCT) on tensor chunks and then selects the Top-k values in the DCT domain. It further employed sign descent on the final aggregated update (Kunstner et al., 2023). Without the DCT transform, this approach can be seen as a special case of SparseLoCo when $H = 1$, the inner optimizer is plain SGD, and the outer optimizer utilizes sign descent. Since the effect of the DCT transform, designed for data with sequential structure where the order of elements matters, is not well understood in this context, and given the additional uncertainty about the role of chunking, in this section we disentangle the contributions of both for DeMo, as well as in the multi-step ($H > 1$) setting. The ablations of these three factors, evaluated purely in terms of loss, are presented in Table 10. Here, the Top-k EF baseline is a simplified DeMo that applies Top-k selection globally to the entire tensor (rather than within chunks) while still utilizing sign descent.

We observe that in the setting with no local steps (Top-k EF, DeMo) the impact of chunking is very significant and the performance of DeMo can be nearly recovered without resorting to the DCT. When using the local setting ($H>1$), we observe that DCT actually degrades performance; however, we also find that the impact of chunking is more limited than in the setting of $H=1$. We hypothesize that chunking and DCT both serve to reduce the effect of outlier values on the scale of individual workers' contributions, which may be less critical in the case of SparseLoCo due to its adaptive inner optimizer.

Notably, DeMo does not have a natural way to incorporate adaptive optimization, and in practice, the sign descent is used to approximate the benefits of the Adam optimizer (Peng et al., 2024). A significant advantage of SparseLoCo is that operating on the pseudo-gradients allows easy integration of adaptive optimizers like Adam in the inner loop.

Table 10: Ablation of tensor chunking and DCT (lower loss is better). We observe that chunking is critical for the performance of DeMo. With $H>1$, DCT degrades performance. All runs use full precision (FP32).

Method	**No DCT**	**DCT**
SparseLoCo ($H > 1$, Chunking, Top-k EF)	**2.72**	2.75
SparseLoCo w/o Chunking ($H > 1$, Top-k EF)	2.73	2.76
DeMo (Chunking, Top-k EF w/ Sign Descent)	2.87	2.83
Top-k EF (w/ Sign Descent)	3.48	2.84
DiLoCo	2.76	–

H Freezing Error Feedback

We apply a short error feedback (OuterEF) freeze at the beginning of training: for the first few outer steps, the error feedback e_r is not utilized. Concretely, during the freeze we neither use nor accumulate in the EF buffer. We find that freezing the OuterEF for the first few outer steps slightly improves training stability and overall performance (see Table 11).

Table 11: Freezing error feedback for the first few outer steps improves training. The final validation loss for 512M models trained with SparseLoCo (3.12% density), R=8 replicas, and communication interval H=15 is reported.

EF Freeze	**Loss**
0%	2.704
5%	2.699

I Error Feedback Decay

We sweep the error feedback decay coefficient $\beta \in \{0.9, 0.95, 0.999\}$ using all other SparseLoCo hyperparameters from Appendix R across different settings. Table 12 reports results for 178M and 512M models at 0.78% and 3.12% density, with $H \in \{15, 50\}$. We find that β=0.95 is a stable choice across all settings.

Table 12: **SparseLoCo is robust to the error feedback decay β.** Final validation loss for different β values, with the remaining SparseLoCo hyperparameters fixed as in Appendix R. Best result in each row is bolded.

Model	Density	H	β=0.9	β=0.95	β=0.999
178M	3.12%	50	**2.99**	**2.99**	**2.99**
178M	0.78%	50	3.09	**3.08**	3.21
512M	3.12%	15	2.71	**2.70**	2.72

J Compression of indices in Top-k

In Top-k methods, the indices of the selected values need to be transmitted alongside the values. When values are aggressively quantized (as in SparseLoCo), this index-transmission overhead becomes significant. In SparseLoCo, we utilize chunk sizes of C=4096, so, naively, we can transmit indices in 12 bits per transmitted value. However, with 2-bit quantization, this overhead becomes significant, motivating further index compression. Assuming a chunk size of C and Top-k selection, we observe that the information-theoretic limit is $\log_2\binom{C}{k}$ bits. For practical cases considered in this work (C=4096 and $k \in \{32, 128, 256\}$), this corresponds to 8.3, 6.3, and 5.3 bits per transmitted value, respectively. In practice, we designed a custom compression algorithm based on sub-chunking and Rice coding that achieves 8.9, 6.6, and 5.6 bits per value for these cases, with negligible runtime overhead.

K Scaling Replicas Across Different Densities and Communication Intervals

We compare DiLoCo and SparseLoCo while varying the number of workers $R \in \{8, 16, 32\}$ and communication intervals $H \in \{15, 50, 100\}$ using 178M and 512M models, and report the final validation loss in Tables 13, 14, and 6. We observe that SparseLoCo outperforms DiLoCo with higher number of parallel workers.

Table 13: Final validation loss for the 178M model while varying the number of workers ($R \in \{8, 16, 32\}$) and the communication interval ($H \in \{15, 50, 100\}$). **Best** is bold.

H=15

Method	Density	Loss	
		R=8	R=32
AdamW	100.00%	2.91	2.91
DiLoCo	100.00%	2.97	3.10
SparseLoCo	0.78%	2.96	3.02
	1.56%	2.93	3.00
	3.12%	**2.91**	**2.99**
	6.25%	2.94	3.00
	12.50%	2.96	3.04
	25.00%	2.96	3.14
	50.00%	2.99	3.29

H=50

Method	Density	Loss	
		R=8	R=32
AdamW	100.00%	2.91	2.91
DiLoCo	100.00%	3.05	3.20
SparseLoCo	0.78%	3.09	3.13
	1.56%	3.03	**3.07**
	3.12%	**2.99**	3.09
	6.25%	**2.98**	3.09
	12.50%	3.00	3.14
	25.00%	3.04	3.25
	50.00%	3.12	3.42

H=100

Method	Density	Loss	
		R=8	R=32
AdamW	100.00%	2.91	2.91
DiLoCo	100.00%	3.12	3.29
SparseLoCo	0.78%	3.20	3.29
	1.56%	3.12	3.26
	3.12%	3.05	3.19
	6.25%	**3.03**	**3.17**
	12.50%	**3.03**	3.21
	25.00%	3.07	3.32
	50.00%	3.17	3.48

L Overtraining Regime with Large Communication Interval

Following Charles et al. (2025), we put SparseLoCo to the test in an overtraining regime by doubling the token budget to 20B and using a larger communication interval of H=250. Our observations are consistent with the trends in Figure 4, and SparseLoCo outperforms DiLoCo in this setting (Table 16).

Table 16: Overtraining on 2× data (20B token budget) with communication interval H=250.

Method	**Density**	**Loss**
DiLoCo	100%	2.77
SparseLoCo	50%	**2.73**
SparseLoCo	25%	2.74
SparseLoCo	12.5%	2.79
SparseLoCo	3.12%	2.97
SparseLoCo	1.56%	3.00

M GPT-2 Experiments

To verify that SparseLoCo applies beyond LLaMA-style models, we also evaluate a 512M-parameter GPT-2 model. We reuse the best hyperparameters from the 512M LLaMA setting for both DiLoCo and SparseLoCo and train with H=15 inner steps on the same dataset and token budget.

Table 14: Final evaluation loss for a 512M model as the number of workers $R \in \{8, 16, 32\}$, communication intervals $H \in \{15, 100\}$, and SparseLoCo's communication densities vary. Best results for each communication interval are presented in **bold**.

Method	Density	Loss (R=8)	Loss (R=16)	Loss (R=32)
AdamW	100.00%	2.69	2.69	2.69
			H=15	
DiLoCo	100.00%	2.76	2.77	2.82
SparseLoCo	0.78%	2.79	2.81	2.84
	1.56%	2.74	2.76	2.79
	3.12%	**2.70**	**2.74**	**2.77**
	6.25%	2.71	2.76	2.78
	12.50%	2.76	2.78	2.82
	25.00%	2.77	2.78	2.93
			H=100	
DiLoCo	100.00%	2.87	2.94	3.05
SparseLoCo	0.78%	3.09	3.14	3.29
	1.56%	3.02	3.06	3.21
	3.12%	2.96	3.03	3.12
	6.25%	2.94	2.97	3.03
	12.50%	2.85	**2.88**	**3.02**
	25.00%	**2.82**	2.89	3.11

Table 15: Model settings for 178M (left) and 2B (right) model scales.

Parameter	**Value**
Total Parameters	177,622,016
Number of Layers	9
Hidden Size	1,024
Intermediate Size	2,688
Attention Heads	8
Vocabulary Size	32,000
FFN Activation	SwiGLU

Parameter	**Value**
Total Parameters	1,972,759,040
Number of Layers	24
Hidden Size	2,560
Intermediate Size	7,680
Attention Heads	20
Key-Value Heads	5
Vocabulary Size	32,000
FFN Activation	SwiGLU

Table 17: **SparseLoCo vs. DiLoCo on a 512M-parameter GPT-2 model.** We report the final validation loss with R=8 replicas.

Method	Density	Loss
SparseLoCo	3.12%	2.89
DiLoCo	100%	2.92

N Equivalence of LOM and Global Outer Momentum

We show that DiLoCo-LOM iterates are actually equivalent to those of DiLoCo.

Proposition 1. *Suppose identical initialization $m_r^{(0)} = m^{(0)} = 0$ for all $r \in [R]$, and fixed outer-momentum coefficient $\beta \in [0, 1)$. Then, for all $t \geq 0$,*

$$\bar{m}^{(t)} = m^{(t)} \quad \textit{and} \quad \bar{\tilde{\Delta}}^{(t)} = \tilde{\Delta}^{(t)},$$

where $\bar{m}^{(t)} := \frac{1}{R}\sum_{r=1}^{R} m_r^{(t)}$ denotes the average of local momentum buffers, $\bar{\tilde{\Delta}}^{(t)} := \frac{1}{R}\sum_{r=1}^{R} \tilde{\Delta}_r^{(t)}$ the averaged LOM Nesterov direction, and $m^{(t)}$ and $\tilde{\Delta}^{(t)}$ the global momentum and Nesterov direction in DiLoCo, respectively. Consequently, the parameter updates of DiLoCo-LOM and DiLoCo are identical at every time step.

Proof. We first show $\bar{m}^{(t)} = m^{(t)}$ by induction, then obtain $\bar{\tilde{\Delta}}^{(t)} = \tilde{\Delta}^{(t)}$ by linearity.

Base case ($t = 0$). With $m_r^{(0)} = 0$ and $m^{(0)} = 0$, we have $\bar{m}^{(0)} = \frac{1}{R}\sum_r m_r^{(0)} = 0 = m^{(0)}$.

Inductive step. Assume $\bar{m}^{(t-1)} = m^{(t-1)}$ for some $t \geq 1$. Averaging the local recursion,

$$\bar{m}^{(t)} = \frac{1}{R}\sum_{r=1}^{R}\left(\beta\, m_r^{(t-1)} + \Delta_r^{(t)}\right) = \beta\left(\frac{1}{R}\sum_r m_r^{(t-1)}\right) + \frac{1}{R}\sum_r \Delta_r^{(t)} = \beta\,\bar{m}^{(t-1)} + \bar{\Delta}^{(t)}.$$

By the global recursion, $m^{(t)} = \beta m^{(t-1)} + \bar{\Delta}^{(t)}$, hence $\bar{m}^{(t)} = m^{(t)}$. For the Nesterov directions,

$$\bar{\tilde{\Delta}}^{(t)} = \frac{1}{R}\sum_{r=1}^{R}\left(\Delta_r^{(t)} + \beta\, m_r^{(t)}\right) = \bar{\Delta}^{(t)} + \beta\left(\frac{1}{R}\sum_r m_r^{(t)}\right) = \bar{\Delta}^{(t)} + \beta\,\bar{m}^{(t)} = \bar{\Delta}^{(t)} + \beta\, m^{(t)} = \tilde{\Delta}^{(t)}.$$

O Sparsity Ablations for DeMo and SparseLoCo

In Figure 7, we compare DeMo and SparseLoCo at different sparsity levels with DiLoCo, and DiLoCo without outer momentum. We train 512M models with R=8 replicas, and for the multi-step baselines, we use a fixed communication interval of H=15.

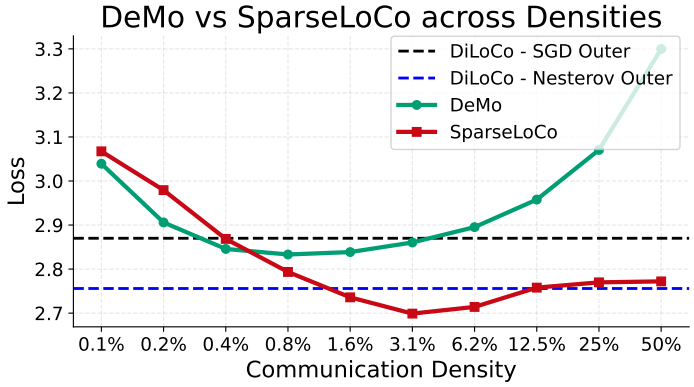


Figure 7: **DeMo and SparseLoCo across varying communication densities.** We compare DeMo and SparseLoCo across varying sparsity levels using the same settings as Figure 4; multi-step methods use a communication interval of H=15.

P EXACT ALGORITHMIC DESCRIPTION OF DILOCO VARIANTS

In this appendix we summarize the DiLoCo variants introduced in this work. All variants share the same local inner loop as DiLoCo: each worker starts from the synchronized parameters $\theta^{(t-1)}$, runs H local AdamW steps, and forms the pseudo-gradient

$$\Delta_r^{(t)} = \theta^{(t-1)} - \theta_r^{(t)}.$$

These methods differ only in the outer step; below we define each variant and provide the corresponding outer-step algorithms. Note that all TOP-k operations are applied chunk-wise as in SparseLoCo.

DiLoCo-LOM. DiLoCo-LOM replaces the global DiLoCo outer momentum with a local outer momentum buffer m_r on each worker. Each worker updates its local momentum using its own pseudo-gradient $\Delta_r^{(t)}$ and forms a local Nesterov direction $\tilde{\Delta}_r^{(t)}$. These directions are then communicated and aggregated across workers to advance the model. Although the momentum buffers are local, the resulting update is exactly equivalent to that of standard DiLoCo by linearity, as shown in Appendix N.

DiLoCo-LOM-Sub-k. DiLoCo-LOM-Sub-k uses the same dense DiLoCo-LOM update, but after the outer update it subtracts the largest-magnitude entries of each local momentum buffer (Line 6 in Algorithm 2). This creates a bridge between DiLoCo and SparseLoCo: communication remains dense as in DiLoCo, while the momentum buffer behaves like an EF residual by removing the entries that would have been selected by TOP-k after each outer step.

DiLoCo+TOP-k with optional EF. DiLoCo+TOP-k compresses each local pseudo-gradient before communication, then applies the standard DiLoCo Nesterov outer optimizer to the aggregated sparse pseudo-gradients. Without EF, workers directly sparsify $\Delta_r^{(t)}$. Workers may also incorporate an optional local EF buffer (Line 2 of Algorithm 3). This is the most direct TOP-k-compressed DiLoCo baseline because only the communicated pseudo-gradient is changed, while the DiLoCo outer optimizer remains unchanged.

DiLoCo-LOM+TOP-k with optional EF. DiLoCo-LOM+TOP-k applies compression after local outer momentum. Each worker first updates its local momentum, forms a local Nesterov direction, and then sparsifies that direction before communication. Workers may also incorporate an optional local EF buffer (Line 4 of Algorithm 4). This provides another natural baseline for combining sparsification with DiLoCo-style outer momentum, discussed in Appendix Q.

Algorithm 2 Outer step for DiLoCo-LOM and DiLoCo-LOM-Sub-k

Require: Pseudo-gradients $\{\Delta_r^{(t)}\}_{r=1}^R$; local momentum buffers $\{m_r^{(t-1)}\}_{r=1}^R$; outer learning rate α; momentum coefficient μ.

1: **for all** $r \in [R]$ **do**
2: $m_r^{(t)} \leftarrow \mu m_r^{(t-1)} + \Delta_r^{(t)}$ // local momenta
3: $\tilde{\Delta}_r^{(t)} \leftarrow \Delta_r^{(t)} + \mu m_r^{(t)}$ // local Nesterov directions
4: $\tilde{\Delta}^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^R \tilde{\Delta}_r^{(t)}$ // aggregate local Nesterov directions
5: $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\tilde{\Delta}^{(t)}$
6: **If DiLoCo-LOM-Sub-k:** // Sub-k: remove top-k values from local momenta
$m_r^{(t)} \leftarrow m_r^{(t)} - \text{TOP-}k(m_r^{(t)})$
7: **end for**

Algorithm 3 Outer step for DiLoCo+TOP-k[+EF]

Require: Pseudo-gradients $\{\Delta_r^{(t)}\}_{r=1}^R$; global momentum $m^{(t-1)}$; optional EF buffers $\{e_r^{(t-1)}\}_{r=1}^R$; outer learning rate α; momentum coefficient μ; EF decay ρ.

for all $r \in [R]$ **do**
  if use EF **then** // Optional EF
    $u_r^{(t)} \leftarrow \rho e_r^{(t-1)} + \Delta_r^{(t)}$
    $\hat{\Delta}_r^{(t)} \leftarrow \text{TOP-}k(u_r^{(t)})$
    $e_r^{(t)} \leftarrow u_r^{(t)} - \hat{\Delta}_r^{(t)}$
  else
    $\hat{\Delta}_r^{(t)} \leftarrow \text{TOP-}k(\Delta_r^{(t)})$ // No EF: sparsify the local pseudo-gradients directly
  end if
  $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^R \hat{\Delta}_r^{(t)}$ // aggregate sparse pseudo-gradients
  $m^{(t)} \leftarrow \mu m^{(t-1)} + \Delta^{(t)}$ // standard DiLoCo momentum
  $\tilde{\Delta}^{(t)} \leftarrow \Delta^{(t)} + \mu m^{(t)}$ // standard DiLoCo Nesterov direction
  $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\tilde{\Delta}^{(t)}$
end for

Algorithm 4 Outer step for DiLoCo-LOM+TOP-k[+EF]

Require: Pseudo-gradients $\{\Delta_r^{(t)}\}_{r=1}^R$; local momentum buffers $\{m_r^{(t-1)}\}_{r=1}^R$; optional EF buffers $\{e_r^{(t-1)}\}_{r=1}^R$; outer learning rate α; momentum coefficient μ; EF decay ρ.

for all $r \in [R]$ **do**
  $m_r^{(t)} \leftarrow \mu m_r^{(t-1)} + \Delta_r^{(t)}$ // local momenta
  $\tilde{\Delta}_r^{(t)} \leftarrow \Delta_r^{(t)} + \mu m_r^{(t)}$ // local Nesterov directions
  if use EF **then** // Optional EF
    $u_r^{(t)} \leftarrow \rho e_r^{(t-1)} + \tilde{\Delta}_r^{(t)}$
    $\hat{\Delta}_r^{(t)} \leftarrow \text{TOP-}k(u_r^{(t)})$
    $e_r^{(t)} \leftarrow u_r^{(t)} - \hat{\Delta}_r^{(t)}$
  else
    $\hat{\Delta}_r^{(t)} \leftarrow \text{TOP-}k(\tilde{\Delta}_r^{(t)})$ // No EF: sparsify the local Nesterov direction directly.
  end if
  $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^R \hat{\Delta}_r^{(t)}$ // aggregate compressed local directions
  $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\Delta^{(t)}$
end for

Q DiLoCo-LOM with Top-k and EF

In Sections 4.1 and 4.3, we study the naive and natural combination of DiLoCo and Top-k+EF sparsification, where we apply the Top-k operation, optionally with EF, to the local pseudo-gradients before communication occurs in the original DiLoCo algorithm (Douillard et al., 2023). This means that the outer momentum is updated from the same aggregated signal on all replicas, so the momentum state remains synchronized across replicas. Since DiLoCo-LOM is equivalent to DiLoCo in the dense case (see Appendix N for the proof), another way to combine Top-k and DiLoCo is to reverse this ordering: first form the local Nesterov directions as in DiLoCo-LOM, and then apply Top-k+EF to those directions before communication. We refer to this variant as DiLoCo-LOM+Top-k+EF.

We evaluate the performance of DiLoCo-LOM+Top-k+EF on a 178M model with R=8, H=15, and 3.12% density. As shown in Table 18, it improves over DiLoCo+Top-k+EF, but still underperforms both dense DiLoCo and SparseLoCo. These results demonstrate that naive combinations of Top-k + EF with DiLoCo and with DiLoCo-LOM are outperformed by SparseLoCo.

Table 18: **Additional DiLoCo-LOM+Top-k+EF ablation.** We report final validation loss for 178M models with R=8 replicas and communication interval H=15. Sparse methods use 3.12% density.

Method	Density	Loss
SparseLoCo	3.12%	**2.91**
DiLoCo	100%	2.97
DiLoCo-LOM+Top-k+EF	3.12%	3.15
DiLoCo+Top-k+EF	3.12%	3.30

R Hyperparameter Selection Details

Table 19 reports the hyperparameter search spaces for the 512M model size. We tune all methods at communication interval H=15 and reuse the best configurations for other settings; general and model-architecture settings are fixed across all runs unless stated otherwise. In Table 15, we provide architectural details for 178M and 2B model sizes. For 178M, we reduce the batch size to 32, leading to an effective batch size of $524,288$, and repeating the hyperparameter sweeps from Table 19, we observe the same optimal settings. For the 2B model size, we increase warmup to 800 and perform a small sweep of learning rates lower than the optimal settings for the 178M and 512M models. Specifically, for DiLoCo we search $\alpha_{\text{inner}} \in \{8\text{e}{-4}, 6\text{e}{-4}\}$ and $\alpha_{\text{outer}} \in \{0.6, 0.4\}$, finding α_{inner}=8e−4, α_{outer}=0.6 optimal; for SparseLoCo we search $\alpha_{\text{inner}} \in \{1\text{e}{-3}, 8\text{e}{-4}\}$ and $\alpha_{\text{outer}} \in \{0.8, 0.6\}$, finding α_{inner}=1e−3, α_{outer}=0.8 optimal. For experiments with $R > 8$ workers, we ensure the same effective batch size used for R=8 by scaling the batch size accordingly. For all training runs we use Chinchilla-optimal (Hoffmann et al., 2022) token budgets unless otherwise stated.

Table 19: Hyperparameter search spaces for the 512M-parameter model scale. Bold entries indicate the best settings. Model and general settings (top) are fixed across all runs. We tune all methods at H=15 and reuse the best hyperparameters when varying H. With a higher number of workers R, DiLoCo's optimal setting remained the same, whereas SparseLoCo enjoys a slightly lower outer learning rate. The effective batch size is given per inner step across all workers.

General Settings	**Value**
Token Budget	10.26B
Effective batch size	4,194,304
Sequence length	2048
Local batch size	256
Workers R	8
Warmup steps	500
Inner gradient clipping	1.0
LR Decay	Cosine
Inner optimizer	AdamW

Parameter	**Value**
Total Parameters	512,398,848
Number of Layers	12
Hidden Size	1536
Intermediate Size	5,440
Attention Heads	12
Vocabulary Size	32,000
FFN Activation	SwiGLU

Setting	**Hyperparameter**	**Search Space**
AdamW Baseline	α	4e-4, 6e-4, 8e-4, 1e-3 **2e-3**, **3e-3**, **4e-3**, 6e-3
DiLoCo - Nesterov Outer	*H=15*, $R \in \{8, 16, 32\}$	
	α_{inner}	4e-4, 6e-4, **8e-4**, 1e-3
	α_{outer}	0.2, 0.4, **0.6**, 0.8, 1.0
	momentum	**0.9**
SparseLoCo (Density=0.78%)	*H=15*, R=8	
	α_{inner}	6e-4, 8e-4, **1e-3**, **2e-3**, 3e-3
	α_{outer}	0.4, 0.6, 0.8, **1.0**
	error-feedback momentum (β)	0.9, **0.95**, 0.999
	H=15, R=16	
	α_{outer}	0.6, **0.8**, 1.0
	H=15, R=32	
	α_{outer}	0.4, **0.6**, 0.8
DiLoCo - SGD Outer	α_{inner}	6e-4, **1e-3**
	α_{outer}	0.8, **1.0**
	momentum	**0.0**
DiLoCo-LOM	α_{inner}	6e-4, **8e-4**, 1e-3
	α_{outer}	0.4, **0.6**, 0.8, 1.0
	momentum	**0.9**
DeMo	α	8e-4, **1e-3**, 3e-3
	error-feedback momentum (β)	0.95, **0.999**

S LIMITATIONS

While we have taken the necessary measures to ensure our current results are sound, there are limitations of our work which naturally arise in empirical studies due to an inability to exhaustively enumerate all relevant experiments. In no particular order, our empirical results (except the large-scale deployment) do not evaluate models larger than 2B parameters, and our study is limited to dense and MoE transformers (e.g., no exploration of more efficient attention variants or hybrid architectures). Regarding theory, we do not prove convergence of SparseLoCo, leaving this to future work.

Due to the cost of LLM pretraining, we do not report error bars or confidence intervals; we instead report ablations across various model sizes, worker counts, communication intervals, and sparsity levels, along with thorough hyperparameter search spaces for baselines.