HETEROGENEOUS LOW-BANDWIDTH PRE-TRAINING OF LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Pre-training large language models (LLMs) increasingly requires distributed compute, yet bandwidth constraints make it difficult to scale beyond well-provisioned datacenters—especially when model parallelism forces frequent, large inter-device communications. We study whether SparseLoCo, a low-communication data parallel method based on infrequent synchronization and sparse pseudo-gradient exchange, can be combined with low-bandwidth pipeline model parallelism via activation and activation-gradient compression. We introduce a heterogeneous distributed training framework where some participants host full replicas on high-bandwidth interconnects, while resource-limited participants are grouped to jointly instantiate a replica using pipeline parallelism with subspace-projected inter-stage communication. To make the recently introduced subspace pipeline compression compatible with SparseLoCo, we study a number of adaptations. Across large-scale language modeling experiments (178M–1B parameters) on standard pre-training corpora, we find that activation compression composes with SparseLoCo at modest cost, while selective (heterogeneous) compression consistently improves the loss–communication tradeoff relative to compressing all replicas—especially at aggressive compression ratios. These results suggest a practical path to incorporating low-bandwidth model parallelism and heterogeneous participants into LLM pre-training.

1 INTRODUCTION

The size and scale of foundation models and large language models (LLMs) pre-training have increased drastically in recent years. This has resulted in the centralization of training into a handful of entities with large enough datacenters. In order to permit distributed training of LLMs that goes beyond a single datacenter, low-bandwidth distributed training for these models has become a topic of increasing interest Douillard et al. (2023); Sarfi et al. (2025); Wang et al. (2023).

A large body of work has focused on techniques for reducing the cost of low-bandwidth data parallelism Reddi et al. (2021). This includes techniques for reducing the frequency of communication Reddi et al. (2021); Stich (2019); Douillard et al. (2023); Thérien et al. (2025), the size of communicated messages Wang et al. (2023); Peng et al. (2024), and the overlap of communication and computation Douillard et al. (2025); Nabli et al. (2024). However, as the size of models grows, the need to train them across multiple accelerators through model parallelism techniques becomes critical. On the other hand, in many cases of interest such as training with a large number of global participants, ensuring a single training replica is contained within a well-interconnected set of accelerators becomes prohibitive. Indeed, in many practical cases, this constraint can significantly restrict the type of hardware that can be used and who can participate (e.g. only those with large resources). To address this question, several recent works consider low-bandwidth model parallelism Ryabinin et al. (2023); Ramasinghe et al. (2025); Singh et al. (2025). Most relevant to our work, Ramasinghe et al. Ramasinghe et al. (2025) have shown that in the setting of standard data parallel methods such as training with the AdamW optimizer, an efficient activation compression method can be combined with pipeline parallelism to significantly reduce the bandwidth constraints of model parallelism.

In this work, we consider the recently introduced data parallel method, SparseLoCo Sarfi et al. (2025), which is able to achieve data parallel performance with drastically reduced communication cost. We ask whether it can be combined with low-bandwidth model parallel techniques, and specifically the

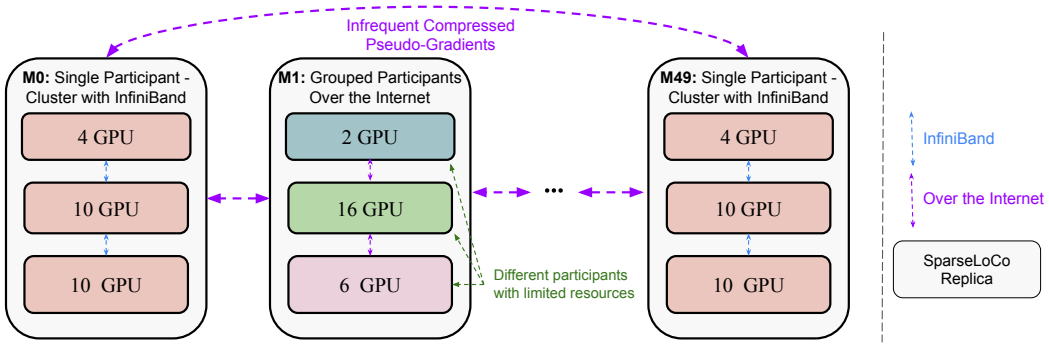

Figure 1: Illustration of heterogeneous training proposed in this work. Across the data parallel axis, we train with M SparseLoCo replicas that perform H local steps before synchronizing using compressed pseudo-gradient communication. A replica can be hosted by a single high-bandwidth cluster (M0 and M49), or by clustering a group of resource-limited participants that collectively form a replica via pipeline parallelism with compressed inter-stage communication (M1). Within each participant, any standard parallelism may be used. This framework allows effective training of large language models with low-bandwidth, heterogeneous resources.

recent approach of Subspace Compression Ramasinghe et al. (2025). We demonstrate the necessary ingredients for Subspace compression required to closely maintain performance of SparseLoCo while achieving the ability to break down model replicas across accelerators with low-bandwidth between them. Subsequently, we propose a heterogeneous scheme in which participants with well interconnected accelerators host full replicas using standard parallelization without subspace projection, while resource-limited participants are grouped to jointly instantiate a replica via pipeline parallelism with subspace-compressed inter-stage communications over the Internet. This enables otherwise unusable nodes to contribute to the training, while applying compression only where bandwidth is the bottleneck.

Our overall contributions are (a) we demonstrate that SparseLoCo—a highly efficient local optimization and gradient compression method—can be combined with model-level activation compression and provide necessary adaptations for this, and (b) we propose a heterogeneous method that applies subspace compression selectively, and we characterize its behavior and scalability, demonstrating it can lead to both a practical use case and better performance.

2 METHOD

We propose a communication-efficient framework for training over *heterogeneous* hardware in distributed training of LLMs which we call Heterogeneous SparseLoCo. Each participant contributes a compute node with heterogeneous resources and communicates with other participants over the Internet, typically under limited-bandwidth. In this setting, (i) participants with well-connected multi-accelerator clusters can host full model replicas, and (ii) limited-compute participants are grouped to jointly instantiate a replica. We then use the recently introduced SparseLoCo Sarfi et al. (2025) to enable infrequent synchronizations and sparse gradient compression across replicas, and we employ the gradient and activation compression method from Subspace Networks Ramasinghe et al. (2025) to reduce within-replica communications of bandwidth-limited participant groups.

Specifically, we consider training a transformer architecture Vaswani et al. (2017); Touvron et al. (2023) that can be partitioned into S transformer pipeline stages. The distributed system is organized into M SparseLoCo replicas, where each replica (m) corresponds either to a single high-bandwidth cluster or to a group of smaller contributors. Within each replica, the model can be parallelized using standard schemes (e.g., tensor parallelism, pipeline parallelism, distributed data parallelism, etc.), and each pipeline stage may itself employ additional parallelism (e.g., tensor parallelism or sharding). This mixed-replica setting is illustrated in Figure 1.

2.1 BACKGROUND: SPARSELOCO

We reduce communication overhead in data parallelism using SparseLoCo Sarfi et al. (2025), which extends DiLoCo Douillard et al. (2023); Reddi et al. (2021) with pseudo-gradient compression and

the removal of outer momentum for an error feedback mechanism. Specifically, we consider H consecutive local optimization steps on each replica before synchronization, reducing communication frequency by a factor of H while performing compression of the pseudo-gradients by selecting topκ largest-magnitude values. For stage s on replica m at outer step t, we perform the following inner step H times:

$$\theta_{s,m}^{(t)} \leftarrow \mathrm{InnerOpt}(\theta_s^{(t-1)}; \mathcal{D}_m), \quad \forall m \in \{1, \ldots, M\} \tag{1}$$

Here, the $\mathrm{InnerOpt}$ is typically the classical AdamW optimizer Kingma & Ba (2015); Loshchilov & Hutter (2019) with the local data being denoted as $\mathcal{D}_m$. The above inner step is repeated H times and subsequently the following global model update is performed:

$$\begin{aligned}
&\Delta_{s,m}^{(t)} \leftarrow \theta_s^{(t-1)} - \theta_{s,m}^{(t)} \\
&e_{s,m}^{(t)} \leftarrow \beta\, e_{s,m}^{(t-1)} + \Delta_{s,m}^{(t)} \\
&\hat{\Delta}_{s,m}^{(t)} \leftarrow Q\left(\left(e_{s,m}^{(t)}\right)\right) \\
&e_{s,m}^{(t+1)} \leftarrow e_{s,m}^{(t)} - \hat{\Delta}_{s,m}^{(t)} \\
&\bar{\Delta}_s^{(t)} \leftarrow \frac{1}{M} \sum_{m=1}^{M} \hat{\Delta}_{s,m}^{(t)}, \quad \theta_s^{(t)} \leftarrow \theta_s^{(t-1)} - \eta\, \bar{\Delta}_s^{(t)}
\end{aligned} \tag{2}$$

The operator retains only the κ largest-magnitude elements per chunk, which are then quantized via $Q(\cdot)$, achieving sparsity while ensuring convergence through error feedback. The error accumulator $e_{s,m}^{(t)}$ preserves discarded pseudo-gradient information across outer steps.

2.2 Background: Communication-Efficient Pipeline Parallelism

With pipeline parallelism, communication occurs at stage boundaries in both the forward and backward passes:

- *Forward pass:* Activation tensors $X_s \in \mathbb{R}^{b \times L \times d_{\text{model}}}$ flow from stage s to stage $s+1$
- *Backward pass:* Gradient tensors $\nabla_{X_s} \mathcal{L} \in \mathbb{R}^{b \times L \times d_{\text{model}}}$ flow from stage $s+1$ to stage s

where d_{model} denotes the hidden dimension, b the micro-batch size, and L the sequence length.

To allow groups of resource-limited participants to form a replica, we adopt the subspace projection method from Subspace Networks Ramasinghe et al. (2025) to compress inter-stage activations and activation-gradients, reducing communication overhead of bandwidth-limited nodes. In this regime, each participant takes on a pipeline stage with over the Internet inter-stage communication.

As shown in Subspace Networks Ramasinghe et al. (2025), transformer activations $X^{(\ell)} \in \mathbb{R}^{b \times L \times d}$ at layer ℓ can be decomposed as:

$$X^{(\ell+1)} = \underbrace{\sum_{i=1}^{\ell} \left(X_{\text{hidden}}^{(i)} W_{p_2}^{(i)} + X_{\text{attn}}^{(i)} W_{p_1}^{(i)} \right)}_{\text{low rank components}} + \underbrace{X^{(0)}}_{\text{high rank components}}$$

where $X^{(0)} = \text{TokenEmbed}(\mathbf{x}) + \text{PosEmbed}$. When the rows of the projection matrices W_{p_1}, W_{p_2} are constrained to the column space of an orthonormal matrix $U \in \mathbb{R}^{d \times k}$ ($k \ll d$), the residual activations $\hat{X}^{(\ell)} = X^{(\ell)} - X^{(0)}$ lie entirely within the k-dimensional subspace $\mathcal{S} = \text{Col}(U)$.

Following the Subspace Networks framework Ramasinghe et al. (2025), we leverage low-dimensional subspace projection to reduce communication overhead between pipeline stages. A randomly initialized matrix $U \in \mathbb{R}^{d \times k}$ is orthonormalized via QR factorization to serve as the projection basis. Since residual activations occupy the low-dimensional subspace spanned by $\text{Col}(U)$, they can be projected without information loss. Additionally, Ramasinghe et al. (2025) suggests to further decompose embedding matrix into fixed high rank ($T_\perp[\mathbf{x}]$) and learnable low rank components,

eliminating the need to transmit embedding matrices across stages. During the forward pass, only the subspace projections are transmitted:

$$\tilde{X}_s = (X_s - T_\perp[\mathbf{x}] - \text{PosEmbed})\, U \in \mathbb{R}^{b \times L \times k}$$

with reconstruction performed via:

$$\hat{X}_s = \tilde{X}_s U^\top + T_\perp[\mathbf{x}] + \text{PosEmbed}$$

For backward gradients, the same projection is applied:

$$(\nabla_{X_s} \mathcal{L})_{\text{compressed}} = \nabla_{X_s} \mathcal{L} \cdot U$$

2.3 Heterogeneous Training Configuration

As shown in Figure 1, we investigate training scenarios involving heterogeneous clusters, where replicas can have accelerators with varying quality. Specifically, we consider environments containing both high-bandwidth interconnects (e.g. InfiniBand clusters) and lower-bandwidth networks (e.g., Internet). We propose to selectively apply pipeline compression based on interconnect bandwidth: well-connected accelerator groups perform standard parallelism without subspace projection, eliminating information loss from compression, while poorly-connected groups employ the Subspace Network Ramasinghe et al. (2025) compression scheme (Section 2.2) to maintain training throughput despite bandwidth constraints. This heterogeneous configuration still operates within the SparseLoCo Sarfi et al. (2025) framework (Section 2.1). The stage level compression decisions apply during the H inner optimization steps, while outer synchronization aggregates updates globally across all worker groups, regardless of their internal communication strategy.

We demonstrate empirically that this heterogeneous strategy, where some replicas do not utilize activation compression, can mitigate performance degradation that arises from activation compression in replicas that use it.

Practically, this allows for large training runs where entire datacenters can act as SparseLoCo replicas, while other replicas aggregate more poorly connected, but lower memory budget accelerators. Notably, accelerators representing different stages can be grouped based on their local proximity or bandwidth. Furthermore, datacenters with limited internal bandwidth (e.g. with cohosted machines) can utilize subspace compression while participating alongside high-bandwidth datacenters.

2.4 Token Embeddings and Heterogeneous Setting

Token embeddings require special care in our mixed setting, where some replicas train with uncompressed parallelization while others apply subspace compression. As previously mentioned in Section 2.2, we decompose the embedding table as $\mathrm{TE} = T_S + T_\perp$, where T_S is constrained to the compression subspace $\mathcal{S}$ and $T_\perp$ stores the high-rank component that is shared among all pipeline stages. In the uniform setting, where all replicas perform subspace compression, T_S is kept in the subspace effortlessly. In contrast, under homogeneous SparseLoCo outer synchronization, averaging across compressed and uncompressed replicas can drive the embedding buffer T_S outside $\mathcal{S}$. We address this with two modifications. First, before training we initialize $T_S \leftarrow \mathrm{TE} \cdot UU^\top$ and $T_\perp \leftarrow \mathrm{TE} - T_S$, where TE is the standard token embedding table from uncompressed replicas. Second, only after each SparseLoCo outer synchronization, we project T_S back to $\mathcal{S}$ while accumulating the projection residual into $T_\perp$:

$$T_\perp \leftarrow T_\perp + \big(T_S - \Pi_{\mathcal{S}}(T_S)\big) \tag{3}$$

$$T_S \leftarrow \Pi_{\mathcal{S}}(T_S)\,. \tag{4}$$

This guarantees $T_S \in \mathcal{S}$ at the start of each inner loop, while $T_\perp$ preserves any out-of-subspace drift, ensuring TE is the same for all replicas after each synchronization.

Analysis of heterogeneous aggregation. Subspace projection introduces systematic bias in the pseudo-gradients. Let Δ^* denote the unbiased pseudo-gradient and $\Delta^*_{\text{proj}} = \Pi_{\mathcal{S}}(\Delta^*)$ the projected variant. The compression bias is $B = \Delta^* - \Delta^*_{\text{proj}}$.

With a fraction α of replicas running uncompressed, the aggregated pseudo-gradient has expected value:

$$\mathbb{E}[\bar{\Delta}_{\text{het}}] = \alpha\Delta^* + (1-\alpha)\Delta^*_{\text{proj}} = \Delta^* - (1-\alpha)B \tag{5}$$

Under uniform compression, the bias is $\|B\|$; under heterogeneous compression, it reduces to $(1-\alpha)\|B\|$. Since $\|B\|$ grows with compression aggressiveness (larger d/k), this predicts the heterogeneous advantage should scale with compression ratio—consistent with Table 2.

Importantly, this bias reduction mechanism requires that uncompressed replicas provide unbiased gradient estimates to anchor the aggregation. Under standard AdamW without local optimization (Table 4), the frequent synchronization prevents compression bias from accumulating across steps, negating the heterogeneous advantage. SparseLoCo's H-step local optimization allows bias to compound, making the correction from uncompressed replicas more valuable.

3 Experiments

In this section, we empirically evaluate the compression schemes proposed in Section 2 through distributed training experiments on practical language modeling benchmarks. Our experimental methodology encompasses systematic ablation studies across model scales, compression ratios, and network configurations to isolate the effects of selective compression. We focus on simulations with equivalent number of steps to determine the validation loss and thereby any performance degradation incurred by Subspace compression. This is in contrast to studying a specific low-bandwidth environment, as in Ramasinghe et al. (2025). This approach allows us to isolate the accuracy performance and communication behavior and simulate it for a variety of bandwidth settings. We establish three principal findings: **(1)** Activation compression between pipeline stages composes with pseudo-gradient sparsification at modest cost, **(2)** Selective compression informed by interconnect bandwidth consistently outperforms compression across all workers , and **(3)** The advantage of heterogeneous configurations grows with compression aggressiveness.

Training Configuration. We train decoder-only transformers at 178M and 512M parameter LLaMA-2 model scales on large-scale pre-training corpora DCLM Li et al. (2024) and C4 Raffel et al. (2020), employing SparseLoCo with $M = 8$ replicas partitioned across 4 pipeline stages. For SparseLoCo, we use the default hyperparameters from Sarfi et al. (2025): a chunk size of 64×64 (i.e., 4096 elements per chunk), TOP-k with k=32 values selected per chunk (corresponding to 0.78% density), error feedback momentum β=0.95, and without quantization. We use AdamW Kingma & Ba (2015); Loshchilov & Hutter (2019) as the inner optimizer, with learning rates of 3×10^{-4} (178M) and 1×10^{-3} (512M). Each replica processes local batches of size 32, yielding an effective global batch size of 524,288 tokens per inner step. Unless otherwise stated, we use $H = 50$ inner steps and Chinchilla-optimal Hoffmann et al. (2022) training budgets on the DCLM dataset Li et al. (2024). Complete architectural specifications, training hyperparameters, and evaluation methodology are detailed in Appendix A.

We vary the subspace projection dimension $k \in \left\{\frac{d}{8}, \frac{d}{12}, \frac{d}{24}, \frac{d}{48}, \frac{d}{96}, \frac{d}{192}, \frac{d}{384}, \frac{d}{768}\right\}$, corresponding to compression ratios, from 87.5% to 99.87%. For the 512M model ($d = 1536$), this translates to subspace dimensions ranging from $k = 192$ to $k = 2$.

We compare three distinct deployment settings:

SparseLoCo Our primary baseline, SparseLoCo with full-precision activations are used across all pipeline stages

SparseLoCo + PP-Compress SparseLoCo with uniform activation compression applied to all worker replicas with the same stage boundaries and compression used for each replica

SparseLoCo + Het-PP-Compress SparseLoCo with Selective compression reflecting heterogeneous interconnect bandwidth. Here, a subset of replicas use pipeline compression while others are uncompressed. This selective approach enables more aggressive compression on bandwidth-constrained links while preserving full-precision communication where network capacity permits, ultimately achieving superior loss-communication tradeoffs compared to the uniform compression strategy.

3.1 Results

SparseLoCo can be practically combined with PP-Compression Table 1 presents our main results on the 512M model. Here, the models are trained with the same number of inner and outer

Table 1: SparseLoCo performance with 178M- and 512M-parameter LLaMA-2 models trained under a compute-optimal token budget. *SparseLoCo* denotes the baseline method without activation compression. *SparseLoCo + PP-Compress* applies subspace-compressed pipeline parallelism with 4 stages to all replicas, while *SparseLoCo + Het-PP-Compress (1/2)* mixes full-precision replicas with compressed ones (half of the workers each). Entries marked with * indicate that hyperparameters were tuned specifically for the compressed setting; unmarked rows reuse the baseline SparseLoCo hyperparameters.

Configuration	Pipeline Compression	178M Loss	178M Perplexity	512M Loss	512M Perplexity
SparseLoCo	0%	3.07	21.50	2.73	15.40
SparseLoCo + PP-Compress	87.5%	3.14	23.00	2.84	17.10
SparseLoCo + PP-Compress*	87.5%	–	–	2.82	16.70
SparseLoCo + Het-PP-Compress (1/2)	87.5%	3.12	22.70	2.82	16.90
SparseLoCo + Het-PP-Compress (1/2)*	87.5%	–	–	2.80	16.50

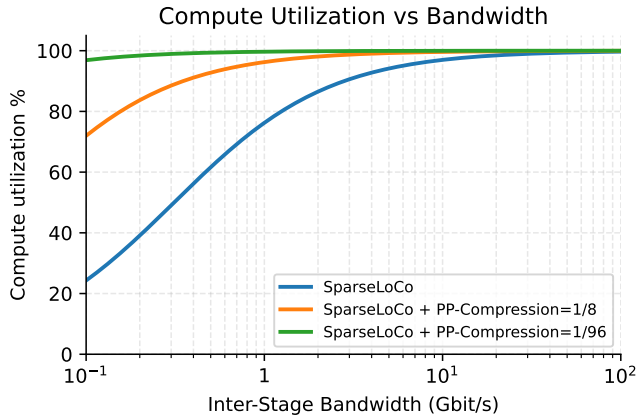

Figure 2: Compute utilization versus inter-stage bandwidth for a 70B-parameter model partitioned into 4 pipeline stages. We compare SparseLoCo with H=50 inner steps under different PP-compression ratios. X-axis denotes the bandwidth between adjacent stages. We observe that PP-compression significantly improves compute utilization in limited-bandwidth regimes.

steps using the different approaches. We observe that (a) using our adapted version of Subspace (**PP-Compress**) compression the performance degradation is minimal compared to **SparseLoCo (Baseline)**, while allowing bandwidth limited pipelining. A modest compression of 87.5% is evaluated between pipeline stages. (b) Furthermore, we can mix replicas using pipeline compression and those without it successfully (**Het-PP-Compress**), while actually decreasing the already small gap to the **SparseLoCo (Baseline)**. Observing Figure 2 which simulates this compression rate we can see that this can have significant impact on wall-clock time. Specifically, we observe that if pipeline replicas operate at even 100 Mb/s – 1 Gb/s links (a practical scenario in over the Internet settings) we can achieve higher than 97% compute utilization while SparseLoCo naively would not support this if stage level links are low-bandwidth.

We also studied the impact of hyperparameter tuning. Note that the SparseLoCo baseline in Table 1 is already well-tuned, we thus show results with Pipeline compression using the same hyperparameters (inner and outer learning rates). On the other hand, we hypothesized that moderate hyperparameter tuning specific to PP-compression can actually improve performance, this is confirmed in the table where we are able to get better performance with tuning in the specific target bandwidth setting (which may be practical in certain scenarios).

Our application of the pipeline compression uses a modest compression compared to Ramasinghe et al. (2025) due to the desire to maintain performance on a per-iteration basis. We study further the effect of the compression level in Table 2 which varies the pipeline compression ratio (k/d) for a 512M model trained with SparseLoCo. At 87.5% compression, heterogeneous configurations gain

Table 2: Performance of 512M models trained with SparseLoCo under different subspace compression ratios. In PP-Compress, all replicas perform PP-compression, and Het-PP-Compress, half of the workers are uncompressed. Heterogeneous setting consistently outperforms uniform PP-Compression, and the benefits grow with higher subspace compression.

		PP-Compress		Het-PP-Compress (1/2)	
k/d	Compression	Loss	Δ (%)	Loss	Δ (%)
1 (SparseLoCo)	0%	2.73	—	2.73	–
1/8	87.5%	2.84	4.0	2.82	3.3
1/24	95.8%	2.89	5.9	2.88	5.5
1/96	99.0%	2.96	8.4	2.94	7.7
1/192	99.5%	3.03	11.0	2.97	8.8
1/384	99.7%	3.04	11.4	2.99	9.5
1/768	99.9%	3.07	12.5	3.00	9.9

Table 3: Final loss for 512M under extended token budgets. Heterogeneous setting effectively mitigates performance degradation arising from activation compression (Compressed runs use k/d=1/8).

Configuration	Token Budget	Loss
SparseLoCo (Baseline)	10B	2.73
SparseLoCo + PP-Compress	12B	2.78
SparseLoCo + Het-PP-Compress (1/2)	12B	2.75

0.50 percentage points over uniform. At 99.9%, where the compression error is much stronger, the heterogeneous setting is more effective with a significantly lower loss degradation. We note that although 10% performance degradation at these very high compression settings is significant, it can in many cases correspond to training the model for a few additional steps. Given the higher compute utilization, this can be possible within a given time. On the other hand, aggressive compression of as much as 99.9% can unlock significant resources at lower per flop costs, accelerating the overall training and potentially reducing its cost.

To validate this, we extend the token budget from 10B to 12B (+20%). Table 3 shows that with 20% additional training flops, heterogeneous compression achieves nearly the same performance as the baseline. Figure 3 simulates the benefits at 1Gb/s cross stage links, compressed training is significantly faster than uncompressed at low-bandwidths between stages.

These results illustrate a general principle: for any bandwidth constraint, there exists a compression ratio $\frac{k}{d}$ that reduces communication overhead below the compute bottleneck. The resulting performance gap is recovered by training on additional tokens—which remains feasible within the same wall-clock budget precisely because of the reduced communication cost. With compression ratios as aggressive as $\frac{k}{d}=\frac{1}{768}$ yielding single-digit degradation (Table 2), this tradeoff is practical across a wide range of bandwidth constraints, from consumer Internet to cross-datacenter links.

Table 4: Normal distributed data parallel (DDP) performance of 512M models trained with AdamW. PP-Compression indicates all replicas performing subspace-compressed PP and in Heterogeneous, only half of the replicas perform compression. Unlike SparseLoCo, we observe heterogeneous setting is in fact detrimental in this setting.

Configuration	k/d	Loss	Δ (%)
AdamW	—	2.75	—
AdamW + PP-Compression	1/8	2.81	2.2
AdamW + Heterogeneous PP-Compression	1/8	2.83	2.9

Heterogeneous advantage in AdamW We also analyze whether the observation of heterogeneous replicas improving performance applies to standard AdamW training as studied in Ramasinghe et al. (2025). Our results are shown in Table 4. We observe that when mimicking per step training,

similar performance degradation as in SparseLoCo is observed, on the other hand the heterogeneous advantage that we have observed does not occur in the case of standard AdamW training.

4 Ablations and Analysis

In this work, we modify Subspace Networks Ramasinghe et al. (2025) by introducing token-embedding adaptation and removing components that are deemed unhelpful under SparseLoCo, namely weight projection and subspace updates using Grassmann manifolds. This section ablates these design choices and reports their impact on training loss. In all cases the same SparseLoCo baseline is used as reported in 1 for the 512M model.

Table 5: Ablations on token-embedding adaptation, weight projection, Grassmann subspace adaptation, and modified AdamW, training a 512M model with SparseLoCo where all replicas perform PP-compression with $\frac{k}{d}=\frac{1}{8}$ compression ratio. We observe that token-embedding adaptation significantly improves the performance, while weight projection, subspace adaptation and modified AdamW are negligible.

(a) Token-embedding adaptation.

TE Adaptation	Loss	Δ (%)
Without	2.89	5.9
With	2.82	3.3

(b) Weight projection.

Configuration	Weight Proj.	Loss	Δ (%)
Uniform	Yes	2.88	5.5
Uniform	No	2.84	4.0
Heterogeneous	Yes	2.87	5.1
Heterogeneous	No	2.82	3.3

(c) Subspace adaptation. A Grassmann update Ramasinghe et al. (2025) is applied every 500 steps.

Configuration	Grassmann	Loss	Δ (%)
Uniform	Yes	2.84	4.0
Uniform	No	2.84	4.0
Heterogeneous	Yes	2.82	3.3
Heterogeneous	No	2.82	3.3

(d) Modified AdamW.

Configuration	Modif. AdamW	Loss	Δ (%)
Uniform	No	2.84	4.0
Uniform	Yes	2.84	4.0
Heterogeneous	No	2.82	3.3
Heterogeneous	Yes	2.82	3.3

Token embedding adaptation. In Section 2.4, we introduce a token-embedding adaptation for the heterogeneous setting, applied after each cross-replica update, to ensure that T_S remains in the subspace $\mathcal{S}$ throughout training. Table 5a (a) shows that this adaptation improves optimization.

Weight Projection Subspace Networks Ramasinghe et al. (2025) projects weight matrices W_{p_1} onto $\mathcal{S}$ after each optimization step. In Table 5 (b), we observe that this is counterproductive under SparseLoCo.

Random Subspace Suffices Subspace Networks Ramasinghe et al. (2025) also proposes adapting the projection basis U via optimization on the Grassmann manifold. Table 5 (c) suggests that this provides no benefit in our setting. A fixed random orthonormal basis yield similar performance, and the subspace does not benefit from dynamic adaptation.

Modified AdamW A modified version of AdamW is used in Ramasinghe et al. (2025), we observe that this does not yield any benefits with SparseLoCo, thus simplifying design choices.

5 Conclusion

In this work, we studied how to combine SparseLoCo Sarfi et al. (2025) with a communication-efficient pipeline parallelism method to enable LLM pre-training under low-bandwidth constraints. We further considered a heterogeneous setting where well-provisioned participants run uncompressed replicas, while resource-limited participants collectively form a replica where they each act as a single stage in the pipeline. To support the heterogeneous setting, we employed subspace-compressed pipeline parallelism Ramasinghe et al. (2025) with modifications for the heterogeneous SparseLoCo training. Across multiple model sizes and datasets, our experiments show that PP-compressed SparseLoCo introduces minimal performance degradation. The heterogeneous training further improves the PP-compressed setting, with larger gains at higher compression rates. Finally, we demonstrate that the benefit of mixing compressed, and uncompressed replicas is specific to SparseLoCo, and is in fact detrimental under standard AdamW training without inner steps.

REFERENCES

Arthur Douillard, Qixuang 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. *CoRR*, abs/2311.08105, 2023. URL https://doi.org/10.48550/arXiv.2311.08105.

Arthur Douillard, Yani Donchev, J Keith Rush, Satyen Kale, Zachary Charles, Gabriel Teston, Zachary Garrett, Jiajun Shen, Ross McIlroy, David Lacey, Alexandre Rame, Arthur Szlam, MarcAurelio Ranzato, and Paul R Barham. Streaming diloco with overlapping communication. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=yYk3zK0X6Q.

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.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Scott Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F. Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M. Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Raghavi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/19e4ea30dded58259665db375885e412-Abstract-Datasets_and_Benchmarks_Track.html.

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

Adel Nabli, Louis Fournier, Pierre Erbacher, Louis Serrano, Eugene Belilovsky, and Edouard Oyallon. Acco: Accumulate while you communicate for communication-overlapped sharded llm training. *arXiv preprint arXiv:2406.02613*, 2024.

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

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

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. Subspace networks: Scaling decentralized training with communication-efficient model parallelism. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=kke9TwtKi0.

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

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. Swarm parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning*, pp. 29416–29440. PMLR, 2023.

Amir Sarfi, Benjamin Thérien, Joel Lidin, and Eugene Belilovsky. Communication efficient llm pre-training with sparseloco. *arXiv preprint arXiv:2508.15706*, 2025.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Vaibhav Singh, Zafir Khalid, Edouard Oyallon, and Eugene Belilovsky. Model parallelism with subnetwork data parallelism. *arXiv preprint arXiv:2507.09029*, 2025.

Sebastian U Stich. Local sgd converges fast and communicates little. In *International Conference on Learning Representations*, 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, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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.

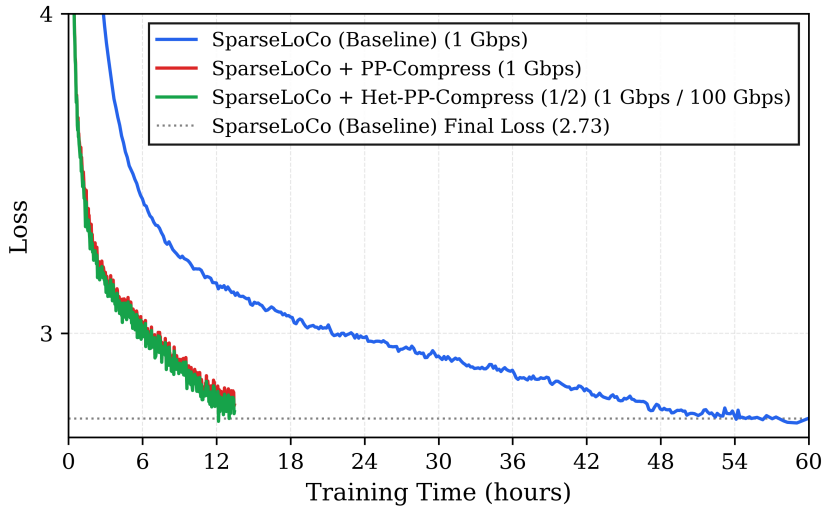

Figure 3: Simulated wall clock times for 1 Gbps inter-stage bandwidth using a 512M model.

A Experiment setup

Model Architecture. We train decoder-only transformers following the LLaMA architecture Touvron et al. (2023) with SwiGLU activations Shazeer (2020) at three scales: 178M, 512M and 1B parameters. Architectural specifications are provided in Table 6.

Table 6: Model architectural configurations and training corpus sizes.

Parameters	Hidden Dim (d)	Layers	Attention Heads	Context Length	Training Tokens
178M	1024	9	8	2048	3B
512M	1536	12	12	2048	10B
1B	1536	16	8	1024	10B

Training loss is the average loss across data parallel replicas, evaluated on unseen data at each outer optimization step. Relative performance degradation is defined as $\Delta = \frac{\mathcal{L}_{\text{compressed}} - \mathcal{L}_{\text{baseline}}}{\mathcal{L}_{\text{baseline}}} \times 100\%$

Scaling to 1B parameters. Table 7 confirms that PP-compression remains effective at the 1B scale, with 87.5% compression yielding a loss of 2.910 compared to the baseline 2.747—a degradation consistent with smaller model scales.

Table 7: 1B parameter results with SparseLoCo on DCLM (sequence length 1024, 10B tokens).

Configuration	Compression	Loss
Baseline	0%	2.747
PP-Compression	87.5%	2.910

Generalization to C4 dataset. Table 8 reports results for a 512M model trained for 10B tokens on the C4 dataset Raffel et al. (2020). The trends mirror those on DCLM: PP-compression introduces modest degradation (3.71%), and the heterogeneous configuration improves over uniform compression (3.32% vs 3.71%). This demonstrates that the benefits of heterogeneous training generalize across pre-training corpora.

Table 8: 512M parameter results with SparseLoCo on C4 (sequence length 2048, 10B tokens)

Configuration	k/d	Loss	Δ (%)
Baseline	—	2.67	—
PP-Compression	1/8	2.77	3.75
Heterogeneous PP-Compression	1/8	2.76	3.37

Table 9: Hyperparameters and model architecture used for training.

Parameter	**Value**
Model size (parameters)	178M / 512M / 1B
Architecture	Decoder-only LLaMa Transformer
Number of layers	9 / 12 / 16
Hidden size (d_{model})	1024 / 1536 / 2048
Number of attention heads	8 / 12 / 8
Head dimension	128
FFN intermediate size	$\{2.63/2.54/4\} \times d_{model}$
FFN Activation	SwiGLU
Vocabulary size	32K
Tokenizer	Byte-Pair Encoding (BPE) / SentencePiece
Context length	2048 / 2048 / 1024 tokens
Batch size (tokens)	512K / 512K / 65K
Training tokens	3B / 10B / 10B
Training steps	5.8K / 19.5K / 156.5K
Learning rate schedule	Cosine decay
Warmup steps	500
Number of replicas (M)	8 / 8 / 2
AdamW Optimizer	
β_1, β_2	0.9, 0.95
Weight decay	0.1
AdamW Baseline ($H = 1$)	
Learning rate	3e-4
SparseLoCo ($H = 50$)	
Inner optimizer	AdamW (see above)
Inner learning rate	1e-3
Gradient clipping	1.0
Outer optimizer	SGD (no momentum)
Outer learning rate	1.0
Compression	
Chunk size	64×64 (4096)
TOP-k per chunk	32
Error feedback momentum (β)	0.95
Outer steps	116 / 391 / 3129
SparseLoCo Tuned (512M model)	
Warmup steps	3910
Outer learning rate	0.8