LEARNED SUBSPACE COMPRESSION FOR COMMUNICATION-EFFICIENT PIPELINE PARALLELISM

Anonymous authors
Paper under double-blind review

ABSTRACT

Pipeline parallelism enables training of large language models (LLMs) that exceed single-device memory but requires frequent communication of activations between pipeline stages. On bandwidth-constrained networks, this communication overhead can become the dominant training bottleneck in decentralized environments. The existing approach, Subspace Networks, mitigates this by constraining activations and model weights to a fixed, shared low-rank subspace across all layers. However, this global constraint restricts the model's representational capacity and leads to substantial performance degradation when trained on the same number of tokens. We propose Manifold Aware Projection Learning (**MAPL**) for communication efficient pipeline parallelism, a framework that formulates inter-stage activation compression as learnable orthogonal projections optimized directly on the Stiefel manifold. Rather than imposing a fixed global subspace, MAPL enables each pipeline stage to dynamically adapt its own compression subspace, coupled with factorized anchor embeddings that restore token-specific offsets with negligible bandwidth overhead. Across LLaMA models from 150M to 3B parameters, MAPL achieves $4\times$ to $11\times$ activation compression with minimal degradation in validation loss relative to uncompressed training. Finally, in a more challenging regime under emulated 1Gbps bandwidth links, MAPL scales to an 8B-parameter model at an extreme $102\times$ activation compression (85× end-to-end) ratio while achieving faster convergence than uncompressed training.

1 INTRODUCTION

Training large language models (LLMs) across geographically distributed, heterogeneous hardware is increasingly attractive (Douillard et al., 2023; Jaghouar et al., 2024; Ramasinghe et al., 2025; Sarfi et al., 2025; Lidin et al., 2026; Nabli et al., 2025; Obeidi et al., 2026), as it enables efficient aggregation of decentralized compute resources. However, this setting places substantial strain on methods designed for centralized clusters (Rajbhandari et al., 2020). Centralized training relies on high-bandwidth, low-latency interconnects, whereas geographically distributed training must operate over commodity wide-area networks with lower bandwidth and higher latency. This communication bottleneck becomes particularly challenging because modern LLMs frequently exceed the memory capacity of a single accelerator, making model parallelism necessary (Shoeybi et al., 2019; Huang et al., 2019; Harlap et al., 2018). Among existing model parallelism strategies, pipeline parallelism (Huang et al., 2019; Harlap et al., 2018) is well suited to memory-constrained training because it partitions sequential layers across devices while requiring relatively lower communication. However, even this communication can become prohibitive over wide-area networks. For every micro-batch, adjacent stages must exchange boundary activations on the forward pass and boundary gradients on the backward pass. On bandwidth-constrained networks, this exchange dominates end-to-end runtime (Ramasinghe et al., 2025).

Therefore, compressing boundary tensors offers a way to reduce communication costs in decentralized pipeline-parallel training. However, this introduces challenges that differ fundamentally from those of gradient compression in data-parallel training (Sarfi et al., 2025; Vogels et al., 2019). In data parallelism, each worker holds a full model replica, and compressed gradients are aggregated across workers before updating the model. In contrast, pipeline-parallel stages hold disjoint portions of the network, so boundary activations directly determine the inputs to subsequent stages. Compression errors in these activations can therefore propagate through the remaining layers, while errors in

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.

boundary gradients can affect the backward signal. Thus, compression methods that work well for data-parallel gradients may not be suitable for pipeline-parallel activations, making boundary tensor compression a substantially harder problem.

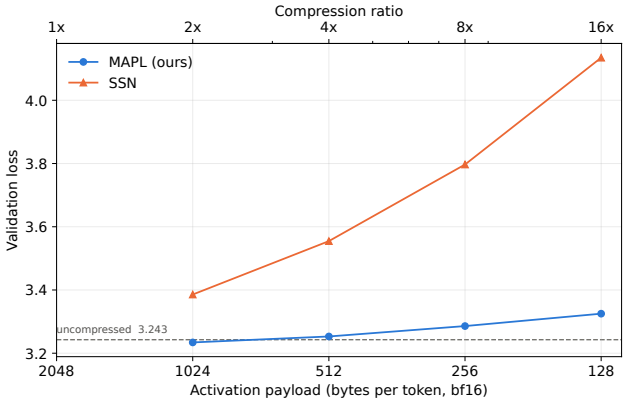


Figure 1: **MAPL achieves pareto frontier at all compression ratios** Validation loss versus activation payload for a 150M-parameter model trained with $P = 4$ pipeline stages on DCLM Li et al. (2024). Our learned projection (MAPL, blue) stays within ≈ 0.08 validation loss of uncompressed training (dashed) across all compression levels, tracing the Pareto frontier.

Recently, Subspace Networks (SSN) (Ramasinghe et al., 2025) proposed imposing a low-rank subspace on the model, motivated by the observation that LLMs converge toward low-rank representations (Janson et al., 2026; Yang et al., 2023; Galanti et al., 2025). SSN uses a single fixed orthonormal basis U_r shared across layers, constrains the transformer output projection weights to this subspace, and transmits boundary activations through their rank-r projection. While this provides a clean compression mechanism, the subspace constraint itself limits the model. A single global subspace forces all stages to communicate through the same geometry, while constraining the weights can reduce the model's representational capacity. As a result, SSN exhibits substantial accuracy degradation relative to an uncompressed baseline trained on the same number of tokens (Table 2).

We argue this constraint is unnecessary. Analyzing the boundary activations of a standard, unconstrained transformer reveals that the low-rank bias targeted by SSN is already naturally present. After removing the token-embeddings, an orthogonal projection onto a subspace of rank $r = d/4$ preserves over 90% of the activation energy at every pipeline boundary as shown in Figure 2. Consequently, low-rank structure in the activations does not need to be structurally imposed and it can be induced directly (See Figure 3).

To this end, we present Manifold-Aware Projection Learning (MAPL), a method for compressing inter-stage activations in pipeline-parallel training that learns a separate compression subspace at each pipeline boundary directly from data, without constraining the transformer weights. Our contributions are threefold. First, we show that optimizing each stage's projector on the Stiefel manifold via manifold-constrained steepest descent (Yang & Lai, 2026) keeps compression and reconstruction isometric, avoiding an instability under which unconstrained projectors underperform even a fixed random projection. Second, because each stage operates in its own learned subspace, we introduce factorized per-stage anchor embeddings that restore stage-specific token offsets at negligible bandwidth cost. Third, we demonstrate that these components together establish a new state of the art in activation compression (Figure 1). Across LLaMA (Touvron et al., 2023a) models from 150M to 3B parameters on 4- and 8-stage pipelines, MAPL attains 4×–11× activation compression with minimal degradation (0.3%–2.4%) in validation loss, closing the gap to uncompressed training. Scaling to an 8B-parameter model on a 4-stage, 1 Gbps emulated pipeline, MAPL achieves faster convergence than uncompressed training at equal wall-clock time under an extreme 102× activation compression ratio (85× end-to-end).

2 PROBLEM SETUP AND BACKGROUND

2.1 PROBLEM SETUP

Notation: We write matrices in bold uppercase, vectors in bold lowercase, and scalars in italics. A decoder-only Transformer (Vaswani et al., 2017; Touvron et al., 2023a) has L layers and hidden dimension d, and is partitioned into P pipeline stages $\mathcal{S}_1, \ldots, \mathcal{S}_P$, giving $P-1$ inter-stage boundaries indexed by $p \in \{1, \ldots, P-1\}$. A micro-batch has batch size B and sequence length T, and the model has vocabulary size V with token embedding table $\mathbf{E} \in \mathbb{R}^{V \times d}$; we write t_{ids} for the token indices of a micro-batch, so $\mathbf{E}[t_{ids}]$ denotes the corresponding embeddings. At boundary p, the forward pass communicates the boundary activation $\mathbf{X}_p \in \mathbb{R}^{B \times T \times d}$ and the backward pass its gradient $\partial \mathcal{L}/\partial \mathbf{X}_p$; we write $\mathbf{X}_{\text{res}} = \mathbf{X}_p - \mathbf{E}[t_{ids}]$ for the residual activation after removing token embeddings. Compression maps $\mathbf{X}_p$ to a rank-r surrogate $\mathbf{Z}_p \in \mathbb{R}^{B \times T \times r}$ with $r < d$, transmitted and reconstructed at the receiver as $\widehat{\mathbf{X}}_p$. We use σ_i for the i-th singular value in descending order and $\mathcal{E}(r) = \sum_{i=1}^{r} \sigma_i^2 / \sum_{i=1}^{d} \sigma_i^2$ for the cumulative energy ratio at retained rank r. We use $||\ ||_F$ to denote frobenius norm and $||\ ||_2$ to denote the spectral norm. For Subspace Networks (SSN), $\mathbf{U}_r \in \mathbb{R}^{d \times r}$ is a fixed shared orthonormal basis of rank r with $\mathbf{U}_r^\top \mathbf{U}_r = \mathbf{I}_r$, $\mathbf{W}_1^\ell$ and $\mathbf{W}_2^\ell$ are the attention and mlp output projection weights at layer ℓ, and PE and $\mathbf{E}_{\text{fixed}} \in \mathbb{R}^{V \times d}$ are the positional and fixed token embeddings. For MAPL, $\mathbf{A}_p \in \mathrm{St}(d, r)$ is the learned per-stage orthogonal projector living on the Stiefel manifold $\mathrm{St}(d, r) = \{\mathbf{A} \in \mathbb{R}^{d \times r} : \mathbf{A}^\top \mathbf{A} = \mathbf{I}_r\}$, $\mathbf{E}_p^{\text{small}} \in \mathbb{R}^{V \times r}$ is the trainable per-stage factorized anchor embedding table paired with a frozen random orthonormal companion projector $\mathbf{P}_p \in \mathbb{R}^{r \times d}$.

Setup: We train a decoder-only Transformer(Vaswani et al., 2017; Touvron et al., 2023a) with a next-token prediction objective under cross-entropy loss $\mathcal{L}$. Each of the P pipeline stages resides on a dedicated device. Stage $\mathcal{S}_1$ holds the token embedding table $\mathbf{E}$ and the earliest layers, and each subsequent stage holds a disjoint block of later layers. On the high-bandwidth interconnects of a centralized cluster, the per-boundary exchange of $\mathbf{X}_p$ and $\partial \mathcal{L}/\partial \mathbf{X}_p$ is negligible. In a decentralized setting, where stages are separated by commodity wide-area links with low bandwidth and high latency, it is not. Therefore, the forward and backward boundary exchanges become the dominant term in end-to-end training time, and reducing them is the central objective of this work.

2.2 BACKGROUND: SUBSPACE NETWORKS

The starting point for compressed pipeline-parallel training is Subspace Networks (SSN) (Ramasinghe et al., 2025). SSN exploits the observation that the attention ($\mathbf{W}_1^\ell$) and output ($\mathbf{W}_2^\ell$) projection weight matrices of Transformers (Vaswani et al., 2017) undergo rank collapse during training, with their effective row spaces converging to a low-dimensional subspace $\mathcal{S} \subset \mathbb{R}^d$ of dimension $r < d$. SSN turns this observation into a compression scheme by *imposing* a single shared subspace on the entire network. A fixed orthonormal basis $\mathbf{U}_r \in \mathbb{R}^{d \times r}$ is constructed once, by QR decomposition of a random matrix, and shared across all layers. The rows of each weight matrix $\mathbf{W}_1^\ell$ and $\mathbf{W}_2^\ell$ are then constrained to $\mathrm{span}(\mathbf{U}_r)$ by projecting the weight matrices into the subspace after every optimizer step. Before compression, the deterministic components of the activations, the positional embeddings PE and the fixed token embeddings $\mathbf{E}_{\text{fixed}}$, are subtracted from $\mathbf{X}_p$. The sending stage then transmits

$$\mathbf{Z}_p = (\mathbf{X}_p - \mathrm{PE} - \mathbf{E}_{\text{fixed}}[t_{ids}])\, \mathbf{U}_r \in \mathbb{R}^{B \times T \times r}, \tag{1}$$

achieving a compression ratio of r/d, and the receiving stage reconstructs

$$\widehat{\mathbf{X}}^p = \mathbf{Z}_p \mathbf{U}_r^\top + \mathrm{PE} + \mathbf{E}_{\text{fixed}}[t_{ids}]. \tag{2}$$

Because the weight confinement guarantees $\mathbf{X}_p - \mathrm{PE} - \mathbf{E}_{\text{fixed}} \in \mathrm{span}(\mathbf{U}_r)$ exactly, the compression in Eq. (1) is lossless *on the constrained model*, though it still incurs a loss relative to an unconstrained baseline. The basis $\mathbf{U}_r$ is refined by Grassmann manifold updates roughly every 500 optimization steps.

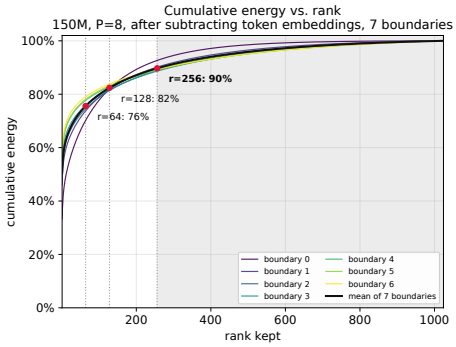


Figure 2: **Cumulative energy vs. rank across pipeline boundaries.** Across all boundaries, representations exhibit extreme low-rank concentration. A rank of $r = 64$ captures 76% of the total energy, while $r = 128$ and $r = 256$ capture 82% and 90%, respectively. The consistent trajectory across every boundary demonstrates that the bulk of activation variance is confined to a low-dimensional subspace throughout the entire pipeline depth.

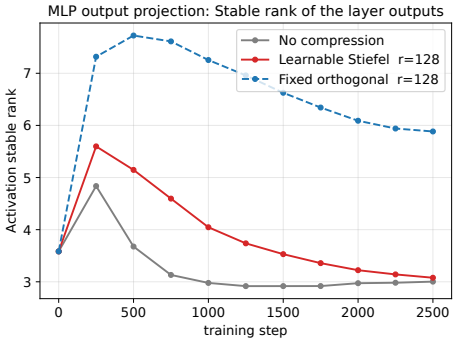


Figure 3: **Dynamics of the stable rank, defined as $\mathrm{srank}(A) = \frac{|A|_F^2}{|A|_2^2}$, for MLP layer output adjacent to the compression boundary throughout training.** MLP output projection matrices have their outputs naturally evolve toward low stable rank when trained with a *learnable* projector (red), approaching values similar to the no-compression baseline. In contrast, a *fixed* orthogonal projector (blue) causes an increase in the stable rank and deviates from no compression training.

3 Manifold Aware Projection Learning

3.1 Motivation: Boundary activations have low rank bias

We analyze a 150M-parameter LLaMA model (Touvron et al., 2023a) trained for 2500 steps on DCLM (Li et al., 2024) with the Muon optimizer (Jordan et al., 2024) at P=8 pipeline stages. Following SSN (Ramasinghe et al., 2025), we subtract the token embeddings and analyze the residual X_{res}, reshaped to $(B{\cdot}T) \times d$, via singular value decomposition (SVD). We quantify spectral concentration through the cumulative energy ratio $\mathcal{E}(r)$. As shown in Figure 2, boundary activations exhibit pronounced low-rank bias across all pipeline stages. A rank of $r = 256$ suffices to retain $\geq$90% of the total activation energy, despite the ambient dimension d=1024 (with mean stable rank ≈ 2.0). The low stable rank reflects a few dominant directions, while the remaining energy decays gradually, so r=256 is needed to retain 90% of the activation energy.

3.2 Learned per-stage projectors

Given that the boundary signal shows a low rank bias, we place at each pipeline boundary p a learnable projector A_p that maps activations into that subspace for transmission and back out at the receiver. Figure 4 shows the overview of our method. Alongside A_p we equip each boundary with a per-stage learnable anchor $E_p^{\text{small}}[t_{ids}]$ and a companion projector P_p that absorb the token-driven offset of the activations (See Appendix A). We do this instead of fixed embedding tables used in SSN (Ramasinghe et al., 2025) as in Equation 1. Since we are using RoPE (Su et al., 2024) positional embedding, we do not need to subtract the PE. Forward compression and reconstruction proceed as

$$Z_p = \left(X_p - E_p^{\text{small}}[t_{ids}]\, P_p\right) A_p \;\in\; \mathbb{R}^{B\times T\times r}, \tag{3}$$

$$\hat{X}_p = Z_p\, A_p^\top + E_{p+1}^{\text{small}}[t_{ids}]\, P_{p+1} \;\in\; \mathbb{R}^{B\times T\times d}. \tag{4}$$

The important design choice is that each boundary learns its *own* A_p from the task loss, with no weight constraint tying the projector to the surrounding layers. We observe in Appendix C that the projector indeed learns to discover the subspace where the activations have high amount of activation energy. In the backward pass we symmetrically route the gradient $\partial\mathcal{L}/\partial X_p$ through the same projector A_p.

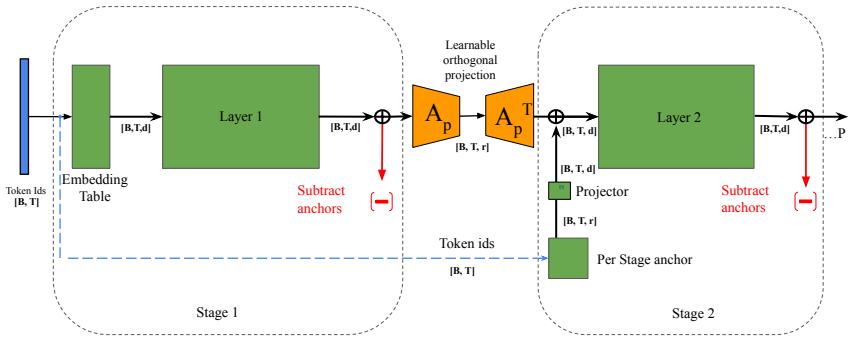


Figure 4: **Overview of MAPL compression at a pipeline stage boundary, repeated across all $P-1$ inter-stage boundaries.** At each boundary, the token-dependent offset is subtracted from the boundary activation $X_p \in \mathbb{R}^{B\times T\times d}$ before transmission (red arrow). The residual activation is then projected to a low-dimensional representation $Z_p \in \mathbb{R}^{B\times T\times r}$ via a learnable orthogonal projector $A_p \in \mathrm{St}(d, r)$, reducing inter-stage communication volume by a factor of d/r. At the receiving stage, $A_p^\top$ reconstructs the full-dimensional activation, to which the destination-stage factorized anchor is added to restore the offset. Integer token IDs are transmitted alongside the compressed activation on the same channel at negligible cost (blue dashed line), enabling each stage to look up its local anchor without additional bandwidth overhead.

3.3 Staying on the Manifold

For A_p to be a valid compressor the two ends of the boundary must agree on a consistent geometry. Therefore, $A_p^\top$ must be a left inverse of A_p, and $A_pA_p^\top$ must be a true orthogonal projection so that projection and reconstruction are isometric on the retained subspace. Learning A_p by ordinary gradient descent does not maintain this as unconstrained updates drive A_p off the set of orthonormal frames, so $A_p^\top A_p$ drifts from the identity and hurts performance to be worse than fixed projection (Appendix D).

We therefore constrain A_p to the Stiefel manifold $\mathrm{St}(d, r)$ throughout training and update it jointly with the model weights using SPEL (Spectral Steepest Descent on the Stiefel Manifold) (Yang & Lai, 2026). At each step, we first project the gradient onto the tangent space of the Stiefel manifold , accumulate this tangent gradient in a momentum buffer, and compute a Muon update direction (Jordan et al., 2024) by orthogonalizing the momentum. The orthogonalized direction does not in general lie in the tangent space, and the resulting step does not preserve orthonormality, so we retract the updated matrix onto $\mathrm{St}(d, r)$ with an additional orthogonalization step. Each update is therefore a Muon-style step driven by the tangent gradient, followed by explicit re-orthogonalization, which keeps $A_p^\top A_p$ close to I_r throughout training (Appendix D). As a result, $A_pA_p^\top$ remains an approximately orthogonal projection onto the learned subspace, so that compression and reconstruction are approximately isometric on that subspace.

Once per optimizer step we sync the gradients of A_p with adjacent stages which adds only $r \times d$ floats (`fp32`) of communication and is modest compared with the activation and gradient traffic and the overhead amortizes with gradient accumulation. We analyze the total communication overhead of our method in Appendix E. Unlike the Grassmann update in SSN which refreshes the subspace only every $\sim$500 steps (Ramasinghe et al., 2025), SPEL updates A_p at every step so that the learned subspace can continuously adapt to the evolving activation geometry. In practice we use a projector learning rate that is an order of magnitude smaller than the weight learning rate ($\times 0.1$). Algorithm 1 summarizes the resulting per-boundary update. Here, $\mathrm{Orthogonalize}()$ denotes the orthogonal polar factor. Specifically, if $M = U\Sigma V^\top$ is its singular value decomposition, then $\mathrm{Orthogonalize}(M) = UV^\top$, which removes the singular-value scaling while preserving the corresponding singular-vector directions. We compute this operation using PolarExpress (Amsel et al., 2026). We provide the complete update rules for all the layers in Appendix G.

Algorithm 1 MAPL: Per-boundary compression and projector update.

Require: Boundary activation X_p, projector $A_p \in \mathrm{St}(d, r)$, momentum buffer m_p, anchor E_p^{small}, companion projector P_p, token IDs t_{ids}, projector learning rate η, momentum β.

1: **Project (sender):** $Z_p \leftarrow \left(X_p - E_p^{\mathrm{small}}[t_{ids}]\, P_p\right) A_p$
2: **Transmit** Z_p and t_{ids}
3: **Reconstruct (receiver):** $\hat{X}_p \leftarrow Z_p\, A_p^\top + E_{p+1}^{\mathrm{small}}[t_{ids}]\, P_{p+1}$
4: **Projector update:**
5: $g \leftarrow \partial \mathcal{L} / \partial A_p$ {$\mathcal{L}$ = Cross entropy loss}
6: $g_R \leftarrow g - \frac{1}{2} A_p \left(A_p^\top g + g^\top A_p\right)$ {Projection to the tangent space}
7: $m_p \leftarrow \beta\, m_p + (1-\beta)\, g_R$ {Momentum update in tangent space}
8: $\Delta \leftarrow \mathrm{Orthogonalize}(m_p)$ {Muon direction from tangent momentum.}
9: $A_p \leftarrow \mathrm{Orthogonalize}(A_p - \eta\, \Delta)$ {Retraction to Stiefel manifold}

Scale	Method	Activation Compression	$P = 4$		$P = 8$	
			Loss ↓	Δ % ↓	Loss ↓	Δ % ↓
150M	Uncompressed	–	3.2427	–	3.2427	–
	SSN (Ramasinghe et al., 2025) (AdamW)	4×	3.6403	12.26	3.6358	12.12
	SSN (Ramasinghe et al., 2025)	4×	3.5546	9.62	3.5742	10.22
	MAPL	4×	**3.2530**	**0.32**	**3.2792**	**1.13**
500M	Uncompressed	–	2.8522	–	2.8522	–
	SSN (Ramasinghe et al., 2025) (AdamW)	6×	3.3354	16.94	3.4435	20.73
	SSN (Ramasinghe et al., 2025)	6×	3.2540	14.09	3.2750	14.82
	MAPL	6×	**2.8737**	**0.75**	**2.9085**	**1.97**
1B	Uncompressed	–	2.7004	–	2.7004	–
	SSN (Ramasinghe et al., 2025) (AdamW)	8×	3.2275	19.52	3.3465	23.93
	SSN (Ramasinghe et al., 2025)	8×	3.1665	17.26	3.1948	18.31
	MAPL	8×	**2.7428**	**1.57**	**2.7649**	**2.39**

Table 1: Final validation loss and relative degradation (Δ %) with respect to the uncompressed baseline across model scales and $P \in \{4, 8\}$. Here we use Muon(Jordan et al., 2024) as the base optimizer for all the methods except the SSN (AdamW). Lower value (↓) is better.

4 Experiments

4.1 Comparison at compute-optimal regime

We first evaluate MAPL on decoder-only transformers based on the LLaMA (Touvron et al., 2023a) architecture at three parameter scales, 150M, 500M, and 1B. All models use a context length of 2048 tokens and are trained with the GPT-NeoX (Black et al., 2022) tokenizer with vocabulary size 50,277 in bf16 precision. We pre-train on the DCLM (Li et al., 2024) corpus and hold out 20M tokens uniformly at random for validation. Following the Chinchilla (Hoffmann et al., 2022) compute-optimal regime, each model is trained on a budget of 20 tokens per parameter. We adopt a hybrid optimizer in which the 2D hidden weight matrices are updated with Muon (Jordan et al., 2024) while embeddings, RMSNorm (Zhang & Sennrich, 2019) gains, the output projection, and the anchor tables are updated with AdamW (Kingma & Ba, 2015; Loshchilov & Hutter, 2019). Training uses a global batch size of 512 with sequence length 2048. We run every method under pipeline-parallel configurations with $P \in 4, 8$ stages.

The uncompressed baseline transmits activations and gradients across pipeline boundaries at the full model width and therefore serves as an upper bound on attainable quality. SSN (Ramasinghe et al., 2025) projects inter-stage tensors onto a fixed low-rank subspace and restricts the output projection weights to the same subspace (Section 2.2). In this setup, since we are not using their modified AdamW update, we project both projection weight matrices to the low rank subspace to attain similar effect. This matches the open source implementation of SSN (Avraham et al., 2025). SSN (AdamW) runs SSN in its originally proposed configuration, which updates all parameters with AdamW. MAPL applies our proposed compression scheme to inter-stage activations and gradients without any weight constraints. Our primary metric is the average cross-entropy validation loss on the held-out DCLM split, together with the relative degradation $\Delta\%$ with respect to the Uncompressed baseline at the same scale.

Table 1 reports the final validation loss after training on the same number of tokens for all methods. MAPL remains close to uncompressed training across every model size and pipeline configuration, with degradation below 2.4% even at the largest scale and highest stage count. SSN incurs substantially larger degradation, exceeding 9% across the evaluated settings. We fix the projector rank at $r = 256$, so the activation compression ratio grows with model size, from $4\times$ for the 150M model to $8\times$ for the 1B model. As the models grow, SSN's degradation climbs to 17–18% while MAPL's degradation increases only modestly, showing that MAPL sustains effective activation compression as the model and compression ratio scale. We attribute this to our method's lack of weight constraints, per stage learned subspace and learned anchors. Running SSN with AdamW in its original configuration consistently hurts it, and the gap widens with scale. Increasing the number of stages from four to eight costs MAPL very little, whereas SSN degrades noticeably, so MAPL stays robust to how finely we split the model.

Scale	P	Method	HellaS.↑	PIQA↑	ARC-E↑	ARC-C↑	Avg↑	Δ ↓
150M	4/8	Uncompressed	29.83	62.24	40.61	23.46	39.04	–
	4	Ours	**29.55**	**61.21**	**37.54**	**23.21**	**37.88**	**-1.16**
		SSN	27.38	58.38	35.98	22.27	36.00	−3.04
		SSN (AdamW)	27.12	57.83	35.69	21.25	35.47	−3.57
	8	Ours	**29.67**	**61.53**	**39.65**	**23.21**	**38.51**	**-0.53**
		SSN	27.38	58.11	34.22	22.27	35.49	−3.55
		SSN (AdamW)	26.67	56.80	34.22	22.44	35.03	−4.01
500M	4/8	Uncompressed	43.41	68.50	50.46	26.37	47.18	–
	4	Ours	**42.27**	**67.36**	**48.48**	**24.15**	**45.56**	**-1.62**
		SSN	31.17	62.46	40.15	23.72	39.37	−7.81
		SSN (AdamW)	29.20	60.55	37.50	22.61	37.47	−9.71
	8	Ours	**40.09**	**66.92**	**47.14**	**25.43**	**44.89**	**-2.29**
		SSN	30.35	61.32	39.06	23.12	38.46	−8.72
		SSN (AdamW)	28.39	59.58	38.55	23.12	37.41	−9.77
1B	4/8	Uncompressed	52.83	71.22	58.21	29.01	52.82	–
	4	Ours	**50.58**	**70.24**	**56.14**	**29.10**	**51.51**	**-1.31**
		SSN	33.93	64.20	41.84	25.77	41.43	−11.39
		SSN (AdamW)	32.14	62.89	42.80	23.21	40.26	−12.56
	8	Ours	**48.88**	**70.62**	**56.36**	**28.41**	**51.07**	**-1.75**
		SSN	33.21	64.85	42.17	24.06	41.07	−11.75
		SSN (AdamW)	29.80	62.24	40.24	24.06	39.09	−13.73

Table 2: **Zero-shot downstream accuracy under pipeline-parallel communication compression.** Results for 150M, 500M, and 1B models trained with P=4 and P=8 pipeline stages. Δ is the drop in Avg relative to the uncompressed baseline of each scale. The best compressed values are shown in bold.

We additionally measure zero-shot downstream accuracy on HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy, and ARC-Challenge (Clark et al., 2018) using `lm-evaluation-harness` (Gao et al., 2024), and report average accuracy across the four tasks in Table 2. The downstream results track the validation loss. MAPL stays within roughly 1–2.3 of the uncompressed average at every scale, while SSN and SSN (AdamW) fall behind by 3–14, with the gap widening as the models grow.

4.2 Scaling Beyond 1B

To test whether this behavior holds past 1B, we pre-train a 3B model for 20B tokens at P=4 under $11\times$ activation compression, using the same optimizer setup as above. Figure 5 shows that MAPL tracks the uncompressed loss curve throughout training and finishes just 1.64% above it. This nearly matches the gap we saw at 1B under lighter compression, even though the model is three times larger and we compress it more aggressively. The cost of MAPL therefore does not compound as we scale, which matters a great deal in decentralized pretraining. We note that this is not close to the Chinchilla (Hoffmann et al., 2022) optimal, but we do a short run due to limited resources to show the similar convergence at 3B scale.

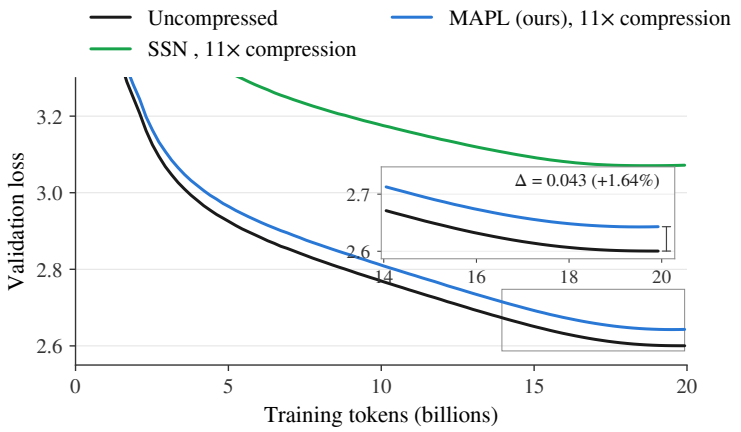


Figure 5: **Validation loss during pretraining of a 3B-parameter model.** We compare standard uncompressed training (black) with MAPL (blue) at $11\times$ compression over 20B training tokens under 4 pipeline stages. MAPL closely tracks the uncompressed baseline throughout training, with the two curves nearly indistinguishable in the early phase. The inset zooms into the final 6B tokens: at the end of training, MAPL reaches a validation loss only 0.043 ($+1.64\%$) above the uncompressed model, showing that MAPL achieves substantial compression with minimal degradation in model quality.

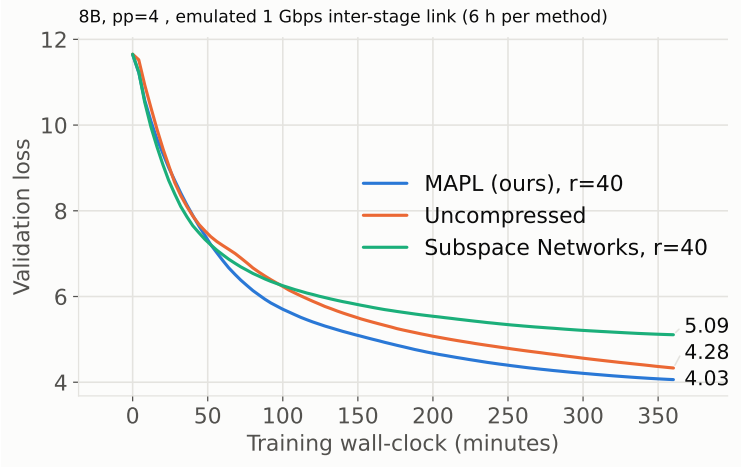


Figure 6: **Validation loss vs. wall-clock training time at extreme activation compression ($\times 102$) for an 8B-parameter model trained with 4-stage pipeline parallelism over an emulated 1 Gbps inter-stage link (bandwidth-constrained setting), with all methods trained for 6 hours.** MAPL (ours, r=40) reaches a lower validation loss at matched wall-clock time than both the uncompressed baseline and Subspace Networks (r=40), converging to 4.03 vs. 4.28 and 5.09 respectively. Because communication is the bottleneck in this regime, MAPL's compressed inter-stage activations translate directly into faster effective training.

4.3 Convergence under bandwidth-limited links

We now evaluate MAPL in the bandwidth-constrained setting comparing under equal wall-clock time. We train an 8B model at $P{=}4$ for six hours per method over an emulated 1 Gbps inter-stage link, using $102\times$ activation compression ($85\times$ end to end). Unlike the token-matched comparisons above, this setup fixes the time budget and lets each method run as far as its communication cost allows. Figure 6 shows the validation loss against the wall clock time for the compared methods. At this link bandwidth the uncompressed run spends most of its time waiting on communication rather than computing. MAPL avoids that bottleneck, pushes more tokens through in the same six hours, and

reaches a lower validation loss than the uncompressed baseline. Compression stops being a quality tradeoff here and wins outright once the network is the bottleneck. SSN has the same communication savings at this rank yet still ends up worse than not compressing at all. We attribute this to the weight constraints which reduces the model's representational capacity. All three runs are still early in training, so we read this as time-to-loss under a fixed budget rather than a statement about final quality.

5 Related works

Low Bandwidth Pre-training. Efforts to democratize large-scale training have studied decentralized optimization over bandwidth-constrained networks, establishing convergence guarantees and communication-efficient variants based on local updates and gossip (Lian et al., 2017; Koloskova et al., 2019; 2020). Communication has been reduced through sparsification (Wang et al., 2017; Wangni et al., 2018; Lin et al., 2018), quantization (Alistarh et al., 2017; Bernstein et al., 2018; Tang et al., 2021; Dettmers et al., 2022), low-rank approximation (Vogels et al., 2019), and error feedback (Wu et al., 2018; Karimireddy et al., 2019). More recent systems demonstrate internet-scale collaborative training, including DeDLOC (Diskin et al., 2021), CocktailSGD (Wang et al., 2023), and DiLoCo-style methods (Douillard et al., 2023; Sarfi et al., 2025; Jaghouar et al., 2024; Douillard et al., 2025). These approaches primarily target data-parallel training, where each accelerator holds a complete model replica. Pipeline-parallel training has different communication dynamics because accelerators hold only parts of the model. SWARM (Ryabinin et al., 2023) studied decentralized pipeline training, while subsequent work explored model parallelism through mixture-of-experts routing (Ryabinin & Gusev, 2020) and reversible architectures (Rivaud et al., 2025). Other approaches reduce pipeline communication by compressing activations (Chen et al., 2021; Bian et al., 2024) or activation changes (Wang et al., 2022). Subspace Networks (Ramasinghe et al., 2025) instead constrain transformer residual streams to a low-rank subspace, using an orthonormal basis to compress boundary activations that is updated infrequently. We instead learn specialized, per-stage projectors without constraining the transformer weights.

Low rank training. A parallel line of work has investigated low-rank geometric structure for memory-efficient gradient compression and model compression, motivated by the empirical observation that deep networks converge to low-rank solutions (LeCun et al., 1989; Yu et al., 2017; Martin & Mahoney, 2021; Yang et al., 2023; Frankle & Carbin, 2019; Galanti et al., 2025). GaLore (Zhao et al., 2024) style methods (Chen et al., 2026; Lialin et al., 2024) projected stochastic gradients onto low-rank subspaces to reduce optimizer state memory, enabling full-parameter pre-training under tight memory budgets. Several works (Janson et al., 2026; Mo et al., 2025; Wei et al., 2024) further explored training low-rank factorized weights to reduce memory and communication requirements. LTE (Huh et al., 2024) used parallel LoRA-style adapters to pre-train models from scratch with reduced memory and bandwidth across parallel accelerators. Pufferfish (Wang et al., 2021) modified the architecture to obtain trained factorized layers that reduce communication costs, but focused on data parallelism. Our work similarly modifies the transformer architecture, but with the complementary goal of enabling efficient inter-stage activation communication for pipeline parallelism.

6 Conclusion

We introduced **MAPL**, a communication-efficient pipeline-parallel training method that learns stage-specific orthogonal compression subspaces directly on the Stiefel manifold. By combining adaptive low-rank projectors with factorized anchor embeddings, MAPL enables aggressive inter-stage activation compression while preserving training quality. Across LLaMA-style models from 150M to 1B parameters and pipeline configurations with 4 and 8 stages, MAPL achieves $4\times$–$8\times$ activation compression while remaining within approximately 0.3%–2.4% of the uncompressed validation loss. At 3B parameters, it finishes within 1.64% of the uncompressed model, while at 8B parameters, MAPL achieves faster convergence than the uncompressed baseline under a wall-clock-matched and bandwidth-constrained setting. MAPL also consistently outperforms Subspace Networks (Ramasinghe et al., 2025) at compared activation compression levels. More broadly, our findings demonstrate that learning a projection provides a practical approach to reducing communication costs in distributed foundation-model training. Our evaluation is limited to models up to 8B parameters and emulated bandwidth settings. An important direction for future work is to evaluate MAPL at larger scales and on real heterogeneous distributed systems and combining it with other orthogonal compression schemes.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, draft portions of the paper, and assist with research ideation and execution. These tools were used to brainstorm and refine research ideas, discuss experimental and implementation choices, and improve the clarity and presentation of the manuscript. We have not used generative AI tools for retrieval and discovery of related work, generation of synthetic datasets, or proving mathematical claims. We reviewed and verified all AI-assisted content, including text and research suggestions, and checked the correctness of any implementation-related assistance through our own analysis and experiments. We take full responsibility for the final content of this work, including all text, claims, methods, and experimental results.

REPRODUCIBILITY STATEMENT

Anonymized source code for MAPL and uncompressed is provided in the supplementary material. The method is fully specified in Section 3 and Algorithm 1, and the complete optimizer step for all parameter groups is given in Appendix G. Appendix F reports all architectures, token budgets, hyperparameters, and the setups of the emulated pipeline experiments. Appendix E gives the communication accounting and cost model. All experiments use the public DCLM (Li et al., 2024) corpus, and downstream evaluation uses the public `lm-evaluation-harness` (Gao et al., 2024).

REFERENCES

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

Noah Amsel, David Persson, Christopher Musco, and Robert M Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. *International Conference on Learning Representations (ICLR)*, 2026.

Gil Avraham, Yan Zuo, Violetta Shevchenko, Hadi Mohaghegh Dolatabadi, Thalaiyasingam Ajanthan, Sameera Ramasinghe, Chamin Hewa Koneputugodage, and Alexander Long. Node0: Model parallel training over the internet with protocol models, 2025. URL `https://github.com/PluralisResearch/node0`.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pp. 560–569. PMLR, 2018.

Song Bian, Dacheng Li, Hongyi Wang, Eric P Xing, and Shivaram Venkataraman. Does compressing activations help model parallel training? *Proceedings of Machine Learning and Systems*, 6: 239–252, 2024.

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

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.

Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pp. 1803–1813. PMLR, 2021.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of LLMs under low-rank constraint? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL `https://openreview.net/forum?id=7aSBAw7tJf`.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=shpkpVXzo3h.

Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Anton Sinitsin, Dmitry Popov, Dmitry V Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, et al. Distributed deep learning in open collaborations. *Advances in Neural Information Processing Systems*, 34:7879–7897, 2021.

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.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Tomer Galanti, Zachary S Siegel, Aparna Gupte, and Tomaso A Poggio. Sgd with weight decay secretly minimizes the ranks of your neural networks. In *The Second Conference on Parsimony and Learning (Proceedings Track)*, 2025.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

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.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Minyoung Huh, Brian Cheung, Jeremy Bernstein, Phillip Isola, and Pulkit Agrawal. Training neural networks from scratch with parallel low-rank adapters. *arXiv preprint arXiv:2402.16828*, 2024.

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

Paul Janson, Edouard Oyallon, and Eugene Belilovsky. Stabilizing native low-rank LLM pretraining. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=kb5wjkqu8q.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://web.archive.org/web/20250122060345/https://kellerjordan.github.io/posts/muon/.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International conference on machine learning*, pp. 3252–3261. PMLR, 2019.

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.

Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International conference on machine learning*, pp. 3478–3487. PMLR, 2019.

Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International conference on machine learning*, pp. 5381–5393. PMLR, 2020.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

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.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReloRA: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=DLJznSp6X3.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

Joel Lidin, Amir Sarfi, Erfan Miahi, Quentin Anthony, Shivam Chauhan, Evangelos Pappas, Benjamin Thérien, Eugene Belilovsky, and Samuel Dare. Covenant-72b: Pre-training a 72b llm with trustless peers over-the-internet. *arXiv preprint arXiv:2603.08163*, 2026.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkhQHMW0W.

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.

Charles H Martin and Michael W Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *Journal of Machine Learning Research*, 22(165):1–73, 2021.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=i0zzO7Hslk.

Adel Nabli, Louis Fournier, Pierre Erbacher, Louis Serrano, Eugene Belilovsky, and Edouard Oyallon. Acco: Accumulate while you communicate for communication-overlapped sharded llm training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Yazan Obeidi, Amir Sarfi, Joel Lidin, Paul Janson, and Eugene Belilovsky. Heterogeneous low-bandwidth pre-training of llms. *arXiv preprint arXiv:2601.02360*, 2026.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 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.

Stephane Rivaud, Louis Fournier, Thomas Pumir, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. PETRA: Parallel end-to-end training with reversible architectures. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=0fhzSFsGUT.

Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. *Advances in Neural Information Processing Systems*, 33:3659–3672, 2020.

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.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 1-bit adam: Communication efficient large-scale training with adam's convergence speed. In *International Conference on Machine Learning*, pp. 10118–10129. PMLR, 2021.

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*, 2023a.

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*, 2023b.

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.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021.

Jialei Wang, Mladen Kolar, Nathan Srebro, and Tong Zhang. Efficient distributed learning with sparsity. In *International conference on machine learning*, pp. 3636–3645. PMLR, 2017.

Jue Wang, Binhang Yuan, Luka Rimanic, Yongjun He, Tri Dao, Beidi Chen, Christopher Ré, and Ce Zhang. Fine-tuning language models over slow networks using activation quantization with guarantees. *Advances in Neural Information Processing Systems*, 35:19215–19230, 2022.

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 *International Conference on Machine Learning*, pp. 36058–36076. PMLR, 2023.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Xiuying Wei, Skander Moalla, Razvan Pascanu, and Caglar Gulcehre. Building on efficient foundations: Effective training of llms with structured feedforward layers. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 4689–4717. Curran Associates, Inc., 2024. doi: 10.52202/079017-0153. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0877af85978e9e630b77f6221db47876-Paper-Conference.pdf.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Jiaxiang Wu, Weidong Huang, Junzhou Huang, and Tong Zhang. Error compensated quantized sgd and its applications to large-scale distributed optimization. In *International conference on machine learning*, pp. 5325–5333. PMLR, 2018.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Kaiwei Yang and Lexiao Lai. Manifold constrained steepest descent. *arXiv preprint arXiv:2601.21487*, 2026.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7370–7379, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

Method	Compression	Val	Δ with baseline	Recovers
Uncompressed baseline	none	**3.2427**	—	0
Manifold learning + No anchor	$r = 256$	3.3325	+2.77%	—
Manifold learning + SSN-style decoupled, static offset	$r = 256$	3.3310	+2.72%	**1.7%**
Manifold learning + Factorized $r = 256$ (ours)	$r = 256$	**3.2792**	+1.13%	**59.6%**
Manifold learning + Full trainable anchor	$r = 256$	3.2491	+0.20%	**93.1%**

Table 3: Ablation of anchor embedding strategies (150M model, $P = 8$, $4\times$ compression). The factorized anchor recovers most of the gap closed by a fully trainable anchor at roughly a quarter of its parameter overhead (12M vs. 51M per stage).

A Factorized Anchors

We follow the finding of SSN (Ramasinghe et al., 2025) that restricting the embedding table to a low-rank manifold causes severe degradation in model performance, which they attribute to a reduction in the representational capacity of the token embeddings. Their solution is a decoupled embedding, where the token embeddings are separated into two parts, one trainable and low-rank ($Rank(\mathbf{E}) = r; r < d$) and the other high-rank but static ($\mathbf{E}_{\text{fixed}}$). Following this finding, we remove the token embeddings from the activation before compression in order to avoid hurting the representational capacity of the model ($\mathbf{X}_{\text{res}} = \mathbf{X}_p - \mathbf{E}[t_{ids}]$). Note that since, we use RoPE (Su et al., 2024) embeddings , we do not need to subtract positional embeddings. We do not, however, adopt the decoupled setup of SSN. In our case we find that using decoupled anchors provides almost no benefit over using no anchors at each stage. We hypothesize that this arises because we apply per-stage projections, which prevents us from using SSN-style token decoupling and from sharing the same subspace for the low-rank components alongside static high-rank components. To validate this, we ablate the anchor formulation on a 150M model across an 8-stage pipeline ($P = 8$, $4\times$ compression).

As shown in Table 3, removing the anchor entirely degrades the validation loss to 3.3325, a $+2.77\%$ degradation relative to the uncompressed baseline. Adding a static frozen embedding projection, analogous to the static offset in SSN, yields only a marginal improvement (3.3310), recovering just 1.7% of the gap. Conversely, using a fully trainable per-stage anchor with a full embedding table to reconstruct token offsets recovers performance to 3.2491 (93.1% of the gap), almost matching the uncompressed baseline (3.2427), but incurs an unacceptable parameter overhead of $V \times d$, which amounts to 51M additional parameters per stage for the 150M model. To strike a better balance between performance and additional parameters, we propose a factorized anchor. This follows the idea of factorizing embedding tables similar to ALBERT (Lan et al., 2020). By learning a low-rank trainable component (E_p^{small}) against a frozen random orthogonal projection (P_p) initialized independently from the first r columns of the Q factor of a QR decomposition of a $d \times d$ Gaussian matrix, this formulation recovers the majority of the performance (3.2792, only $+1.13\%$ degradation, 59.6% of the gap) while adding a small parameter overhead to the pipeline stages, amounting to 12M additional parameters per stage. This overhead can be significant at small scales, but as we show for larger model sizes such as 3B and 8B, we observe the benefits while it remains a fixed 12M additional parameters per stage and thus becomes far less significant.

A.1 Anchors Compensate for Compression Rather Than Providing a Generic Capacity Boost

A natural concern is that the additional parameters introduced by the anchors provide a significant boost to representational capacity on their own, which would make the comparison against the uncompressed baseline less fair. To test this, we add the same anchor mechanism to an uncompressed model. We run this control at 150M with $P = 8$, the worst case in our setting, and use the DCLM dataset (Li et al., 2024).

The anchors do not act as a generic capacity boost. We add an identically configured per-stage factorized anchor ($r = 256$, with the same factorization and parameter budget), trained end to end from scratch, to an uncompressed model. This changes the loss by only 0.64%, a smaller change. The same mechanism reduces the loss of the compressed model by 1.60% (from 3.3325 to 3.2792). The two runs use identically configured anchors that are learned independently. If the added anchor

Setting	Compression	Anchor	Val	Δ vs control
Uncompressed control	identity (lossless)	—	3.2427	—
Uncompressed + factorized anchor	identity	trainable r=256	3.2218	−0.64%
Manifold learning + No anchor	r=256	—	3.3325	+2.77%
MAPL	r=256	trainable r=256	3.2792	+1.13%
+ static offset (SSN style)	r=256	frozen (0 trainable params)	3.3310	+2.72%
+ rank-64 anchors	r=256	trainable r=64	3.3190	+2.35%
+ full trainable anchors	r=256	full trainable	3.2491	+0.20%

Table 4: Anchor control at 150M, $P = 8$. Δ is the relative change in validation loss with respect to the uncompressed control. Adding the same anchor mechanism to an uncompressed model yields only a small change, whereas it substantially helps the compressed model.

capacity were simply a generic booster, it would help both settings comparably. It does not, so the benefit is specific to compensating for compression. A frozen anchor with no trainable parameters does not recover a meaningful amount, indicating that the anchor must be learned. A four times smaller rank-64 anchor recovers about 25% of the gain, while a fully trainable anchor goes further and brings the loss to within 0.2% of the uncompressed control. Anchor capacity therefore matters for compensating compression, even though the same capacity yields little benefit without compression.

A related concern is that MAPL's advantage over SSN stems from its anchor parameters rather than from removing the weight constraint and learning projector. To test this, we equip SSN with the same factorized anchor ($r = 256$) at 150M with $P = 4$, keeping SSN's low-rank weight constraint and Muon optimizer unchanged. Table 5 shows that the anchor improves SSN by only 0.018. This shows that SSN's performance degradation is mainly caused by its restrictive weight constraints which cannot be recovered by having a learnable anchor embeddings.

Method	Weights	Anchor	Val	Δ vs uncompressed
Uncompressed	unconstrained	—	3.2427	—
SSN	low-rank constrained	SSN fixed offset	3.5546	+9.62%
SSN + factorized anchor	low-rank constrained	trainable r=256	3.5363	+9.05%
MAPL	unconstrained	trainable r=256	**3.2530**	**+0.32%**

Table 5: Equipping SSN with MAPL's factorized anchor at 150M, $P = 4$, and $r = 256$ (4× activation compression). Adding a trainable factorized anchor provides little benefit to SSN, where the model weights are constrained to the compression subspace. In contrast, the factorized anchor is effective in MAPL, where the model weights remain unconstrained, suggesting that its benefit comes from allowing the model to retain information outside the learned compression subspace.

B Extended Background: Subspace networks

SSN (Ramasinghe et al., 2025) exploits a structural property of trained Transformers: the output projection weights $W_2^\ell \in \mathbb{R}^{d_{\text{ff}} \times d}$ and attention projection weights $W_1^\ell \in \mathbb{R}^{d \times d}$ undergo *rank collapse* during training, with their effective row space converging to a low-dimensional subspace $\mathcal{S} \subset \mathbb{R}^d$ of dimension $r < d$. This collapse arises naturally in Transformer (Vaswani et al., 2017) training.

SSN operationalizes this observation via five steps. **(1)** A fixed orthonormal basis $U_r \in \mathbb{R}^{d \times r}$ is constructed by QR decomposition of a random matrix. **(2)** The rows of each W_2^ℓ and W_1^ℓ are constrained to $\text{span}(U_r)$ by projecting the weights to the subspace or by replacing the per-parameter adaptive scaling in AdamW with a row-wise constant variant. **(3)** Deterministic components—positional embeddings PE and fixed token embeddings E_{fixed}—are subtracted from X_p before compression. **(4)** The stage transmits the compressed activation

$$Z_p = \big(X_p - \text{PE} - E_{\text{fixed}}\big)\, U_r \;\in\; \mathbb{R}^{B \times T \times r}, \tag{5}$$

achieving a compression ratio of r/d. **(5)** The receiving stage reconstructs $\hat{X}_p = Z_p U_r^\top + \text{PE} + E_{\text{fixed}}$.

Because weight confinement ensures $X_p - \text{PE} - E_{\text{fixed}} \in \text{span}(U_r)$ exactly, SSN's compression is *architecturally lossless* on its modified model, though it incurs loss relative to an unconstrained baseline. The basis U_r is refined via Grassmann manifold updates every $\sim$500 steps.

Limitations. SSN's losslessness is contingent on strict weight confinement. The subspace is fixed at initialization and only coarsely adapted; the model is forced to conform to a predetermined geometry rather than inducing a task-optimal one. In §3.1, we showed that pipeline boundary activations exhibit low-rank bias *naturally*. Even when trained with an optimizer with full rank updates such as Muon (Jordan et al., 2024) without any architectural constraint. Table 6 compares the difference between our method and SSN.

Feature	Subspace Networks (SSN)	MAPL (Ours)
Unconstrained Full-Rank Weights	×	✓
Learnable Projection Basis	Slow grassmann updates	✓
Decoupled fixed low rank Embeddings	✓	×
Learnable Per-Stage Anchor	×	✓

Table 6: Comparison of architectural features between Subspace Networks (SSN) and MAPL. Our method enables high-ratio compression without constraining model weights to low-rank subspaces.

C Analysis: what the method learns

Figure 7 shows three properties of the learned projectors and explain why adaptive projection outperforms a fixed baseline. First, each boundary learn a distinct subspace. Off-diagonal principal angles between stages range from 53° to 72°, where adjacent stages share the most overlap and distant stages approach orthogonality. This alignment matches the residual stream's shift from lexical to task-specific representations with depth (Zeiler & Fergus, 2014), justifying an individualized A_p at every boundary. Second, the learned projector captures far more energy than an equal-rank fixed basis, achieving roughly 80% within $1{,}500$ steps versus the fixed baseline's 36% ceiling. Because the preceding MLP outputs adapt directly to A_p (Section 3.1), the model actively shapes the low-rank structure rather than merely exploiting existing redundancy. Finally, the projection preserves pairwise token geometry without explicit constraints: projected cosine similarities mirror full-dimensional values at Pearson $r = 0.992$, confirming that the compressed signal preserves the relational features downstream layers rely on.

D Keeping Projectors in the Stiefel Manifold

In this section, we demonstrate the necessity of constraining subspace projectors to the Stiefel manifold during training. If the subspace projectors are trained naively, standard optimizer updates cause them to deviate from the manifold, thereby violating the orthonormality constraint. Consequently, the projection loses its isometric properties, allowing the model to learn representations outside the intended subspace.

To empirically validate and isolate this, we train a 150M parameter model under three distinct projection configurations without anchor offsets. In the baseline setup, we initialize an orthogonal projector and freeze it for the duration of training. In the second configuration, the projector is updated using standard Euclidean gradients without any manifold constraints using Muon as the optimizer Jordan et al. (2024). As anticipated, these unconstrained updates cause the weight matrix to escape the manifold and lose its orthogonality, which actively degrades performance—yielding worse results than simply utilizing a fixed projection. Finally, enforcing the manifold constraint via SPEL (Yang & Lai, 2026) updates resolves this issue and achieves the lowest validation loss. These results are summarized in Table 7.

E Communication Analysis

MAPL introduces two sources of communication in addition to the compressed activations, both of which the compression scheme needs. First, as in subspace networks (SSN), each stage must receive the token ids of the micro-batch. Second, when the projector is updated (once per optimizer step), the two stages adjacent to boundary p exchange their partial gradients of A_p so that their fp32 master copies apply the same update: one $d \times r$ fp32 message in each direction, i.e. $8\,d\,r$ bytes per boundary

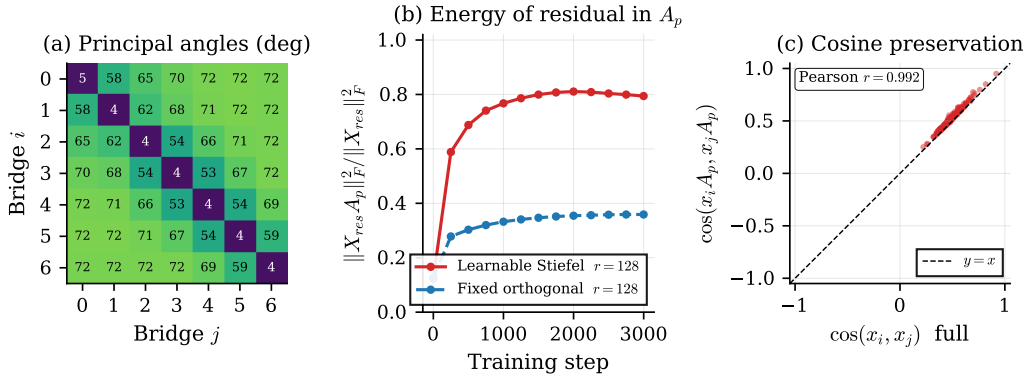


Figure 7: **Learned orthogonal projectors learn distinct subspaces and carry more energy in the subspace.** (a) Pairwise mean principal angles (degrees) between learned Stiefel projectors across pipeline stages i and j, computed via $\arccos(\sigma_k(A_i^\top A_j))$. Large off-diagonal angles (up to 72°) confirm that projectors across non-adjacent stages converge to geometrically distinct subspaces, while near-zero diagonal values verify self-consistency. (b) Frobenius-norm energy of residual activations preserved under projection, $\|X_{\text{res}}A_p\|_F^2/\|X_{\text{res}}\|_F^2$, as a function of training step for learned Stiefel (r=128, red) versus fixed orthogonal (r=128, blue) projectors. Learned projectors retain approximately 2.2× more residual energy (~ 0.80 vs. ~ 0.36). (c) Cosine similarity preservation between token pairs: $\cos(x_iA_p,\, x_jA_p)$ versus $\cos(x_i, x_j)$ on the full-dimensional representations. A Pearson correlation of r=0.992 confirms that A_p faithfully preserves the pairwise relational geometry of the activation space, closely tracking the identity line (y=x).

Projector update	**Val loss**	**Delta**	$\|A_p^\top A_p - I_r\|_F$
Uncompressed	3.2427	—	—
Fixed orthogonal projector	3.2800	+1.15%	2.4×10^{-3}
Learnable projector + Muon	3.3018	+1.82%	3.3×10^{2}
Learnable projector + SPEL updates	**3.2738**	**+0.96%**	6.6×10^{-2}

Table 7: **Impact of Stiefel manifold constraints at 150M, P=4, r=256.** Unconstrained Muon updates degrade performance below that of a fixed projection, while SPEL updates optimize the projector while preserving orthogonality. $\|A_p^\top A_p - I_r\|_F$ is the final value, averaged over the $P-1$ boundaries; at initialization every arm sits at 2.4×10^{-3}. Deltas are relative to the uncompressed run. All runs without anchor offsets showing isolated effect of learning the projection.

per optimizer step, independent of the batch. We begin by quantifying these costs relative to the compressed activation traffic, which scales as $B \times T \times r$, across model widths (Section E.1). We then quantify the wall-clock effect of MAPL's payload reduction in pipeline-parallel (PP) training. We measure per-step compute and transmitted bytes on real GPUs and combine them with an analytic model of the inter-stage link to obtain throughput and break-even bandwidths. All communication volumes reported in this appendix include the token-id and projector traffic.

E.1 Auxiliary communication

Let B be the micro-batch size, T the sequence length, m the number of micro-batches per optimizer step, and r the code dimension. Per boundary and per optimizer step, the three communication terms are:

- **Compressed activations** (bf16, forward activation and backward gradient): $4\,m\,B\,T\,r$ bytes.
- **Token ids** (int32, forward only): $4\,m\,B\,T$ bytes.
- **Projector gradient exchange** (fp32, once per optimizer step): $8\,d\,r$ bytes, i.e. one $d \times r$ fp32 message in each direction. Both endpoint stages keep an fp32 master copy of A_p and

apply the same deterministic SPEL update, so A_p itself is never sent, but the gradient of the tied projector splits across the link.

Relative to the compressed activations, the token-id overhead is $1/r$, which is 0.39% at $r{=}256$ and does not depend on model width. The projector overhead is

$$\frac{8\,d\,r}{4\,m\,B\,T\,r} = \frac{2\,d}{m\,B\,T}. \tag{6}$$

This ratio is independent of r, grows linearly with width d, and is amortized over the m micro-batches of an optimizer step. Table 8 evaluates it for $B{=}1$ and $T{=}2048$. The measurements in the rest of this appendix use one micro-batch per optimizer step and update the projector at every step, which corresponds to the $m{=}1$ column. They are therefore the worst case setting for MAPL; with gradient accumulation over $m > 1$ micro-batches the projector cost shrinks proportionally. At the global batch of the token-matched runs ($m\,B\,T = 512 \times 2048$ tokens) it is 0.20% (150M), 0.29% (500M), 0.39% (1B) and 0.54% (3B); at the small batch of the 8B wall-clock experiment (32×2048 tokens) it is 12.5% for a pairwise exchange.

Width d	Exchange size	$m{=}1$	$m{=}8$	$m{=}32$
2048	4.19 MB	200.0%	25.0%	6.3%
4096	8.39 MB	400.0%	50.0%	12.5%
8192	16.78 MB	800.0%	100.0%	25.0%

Table 8: Projector gradient-exchange traffic (fp32, $8\,d\,r$ bytes per boundary per optimizer step counting both directions, $r{=}256$) as a fraction of compressed activation traffic (Eq. 6), for $B{=}1$ and $T{=}2048$. Token-id traffic adds 0.39% in every setting.

Setup. All runs use $pp{=}4$ (three stage boundaries), sequence length 2048, bf16, and one micro-batch per optimizer step ($m{=}1$). Per-step compute (forward and backward, including compression, decompression and anchoring) is measured on NVIDIA RTX A6000 GPUs. Activation bytes per token are read from communication counters and include both the forward activation and the backward gradient at each boundary; token-id and projector traffic are added using the expressions above. We report the 1B model ($d{=}2048$) at micro-batch sizes 1 and 2 and the 8B model ($d{=}4096$) at micro-batch size 1, with the architectures of Appendix F and $r{=}256$ for both SSN and MAPL.

Cost model. We model the step time as

$$T_{\text{step}} = T_{\text{comp}} + \frac{V}{B}, \tag{7}$$

where T_{comp} is the measured compute time, V is the total bytes transmitted per step (summed over boundaries), and B is the link bandwidth. This model assumes communication is fully exposed, i.e. not overlapped with compute. A compressed method is faster than the uncompressed baseline whenever $B < B^*$, with the break-even bandwidth

$$B^* = \frac{V_{\text{unc}} - V_{\text{comp}}}{T_{\text{comp}}^{\text{comp}} - T_{\text{comp}}^{\text{unc}}}. \tag{8}$$

E.2 Activation payload per token

The activation payload per token per boundary depends only on the dimension of what is transmitted. The uncompressed baseline sends the full d-dimensional activation, so its payload grows with hidden size. SSN and MAPL send a fixed-rank code ($r{=}256$), so their activation payload is constant across scale, and the activation compression ratio grows with model size (Table 9). Table 9 excludes the per-step token-id and projector traffic, which Table 10 includes.

E.3 Measured compute and communication

Table 10 reports measured compute and the total per-step communication volume, including activations, token ids, and projector updates, together with the break-even bandwidth from Eq. (8).

Method	Transmitted	1B (d=2048)	8B (d=4096)
Uncompressed	d, bf16	8192 B	16384 B
SSN / MAPL	r=256, bf16	1024 B (8×)	1024 B (16×)

Table 9: Activation payload per token per boundary (forward activation + backward gradient), excluding token-id and projector traffic. Ratios are relative to the uncompressed baseline.

Including the projector exchange lowers MAPL's end-to-end compression ratio from the activation-only values in Table 9 to 2.7–4.0× at m=1 but this is the worst case analysis and in real world higher number of micro batches are used.

Config	Method	Compute (ms)	Δ (ms)	Comm (MB/step)	B^* (Gbps)
1B, mbs 1 (2048 tok)	Uncompressed	232.0	–	50.33	–
	SSN	240.9	+8.9	6.32 (8.0×)	39.4
	MAPL	242.9	+10.9 (4.7%)	18.90 (2.7×)	23.1
1B, mbs 2 (4096 tok)	Uncompressed	480.7	–	100.66	–
	SSN	489.2	+8.5	12.63 (8.0×)	83.2
	MAPL	493.6	+12.9 (2.7%)	25.22 (4.0×)	46.8
8B, mbs 1 (2048 tok)	Uncompressed	1177.9	–	100.66	–
	SSN	1196.6	+18.7	6.32 (15.9×)	40.5
	MAPL	1199.6	+21.7 (1.8%)	31.48 (3.2×)	25.6

Table 10: Per-step compute (measured) and total communication volume (activations, token ids, and the fp32 projector gradient exchange, m=1). Δ is the compute overhead relative to the uncompressed baseline. B^* is the break-even bandwidth (Eq. 8).

E.4 Wall-clock speedup

Table 11 gives the speedup of MAPL over uncompressed PP under Eq. (7) at three link bandwidths, including all auxiliary communication.

Config	100 Mbps	1 Gbps	10 Gbps
1B, mbs 1	2.43×	1.61×	1.06×
1B, mbs 2	3.40×	1.85×	1.09×
8B, mbs 1	2.48×	1.37×	1.03×

Table 11: Emulated speedup of MAPL over uncompressed PP, including token-id and fp32 projector-exchange traffic (m=1).

Two scale effects act in opposite directions. First, MAPL's compute overhead shrinks as a fraction of the step as the model grows (4.7%, 2.7%, and 1.8%), because the compression and anchoring cost is small relative to the transformer FLOPs at larger scale. Second, at faster links the speedup is smaller at 8B (1.37× versus 1.61–1.85× at 1 Gbps), because the 8B step is more compute-bound and exposed communication is a smaller share of it. In the bandwidth-bound regime MAPL targets ($\leq$ 1 Gbps), MAPL is 1.4–3.4× faster at every scale tested, even when the projector gradient is exchanged in fp32, in both directions, at every step.

E.5 Break-even bandwidth

Including all auxiliary communication, MAPL transmits 2.7–4.0× fewer bytes than uncompressed PP at a compute overhead of 1.8–4.7%. At the operating points above, it is therefore faster than uncompressed PP on any inter-stage link slower than approximately 23–47 Gbps, which covers commodity Ethernet, cloud, and wide-area interconnects. On InfiniBand- or NVLink-class links

the uncompressed baseline is faster, as expected. These figures use one micro-batch per optimizer step which is the worst case analysis, but with gradient accumulation the projector cost is amortized (Table 8) and MAPL's end-to-end compression approaches the activation-only ratios of Table 9.

E.6 Limitations of this analysis

The link is emulated analytically on a single node rather than measured on a throttled multi-node deployment. Compute times and activation byte volumes are measured, token-id and projector volumes are computed from their sizes, and the transfer time is computed from Eq. (7). Because the model assumes fully exposed communication, it is an upper bound on the speedup achievable when a schedule overlaps communication with compute. Validating these results with real bandwidth-limited multi-node NCCL runs is left to future work.

F Implementation Details

In this section, we provide a comprehensive overview of the models, datasets, and hyperparameters used across all experiments. Our goal is to ensure full reproducibility of the pipeline-parallel compression results presented in the main text. All token-matched experiments (Tables 1 and 2, Figures 1 and 5, and all ablations) were run on nodes with 4× NVIDIA H100 80 GB GPUs (a few runs on 8×H100 nodes), used as data-parallel replicas. In these runs pipeline parallelism is *simulated*: every replica holds the full model on one GPU, the P stages are executed in sequence, and the boundary activations and gradients pass through the compression modules exactly as they would over a link, with the wire payload accounted analytically (bytes per token at bf16). Gradients are accumulated over micro-batches of 4 sequences and averaged across replicas with one all-reduce per optimizer step. Only the 8B wall-clock experiment (Figure 6) uses real pipeline placement; its setup is described in Appendix F.5. We use PyTorch (Paszke et al., 2019) 2.9 with the HuggingFace transformers (Wolf et al., 2019) 5.5 implementation of Llama 2 (Touvron et al., 2023b) (RoPE (Su et al., 2024) positional embeddings, RMSNorm, SwiGLU, no biases), with the Liger-Kernel fused RoPE/RMSNorm/SwiGLU kernels and PyTorch SDPA attention (flash path), Python $\geq$3.10. Training runs under bf16 autocast with fp32 master weights and optimizer state.

F.1 Model Architectures

All experiments evaluate decoder-only transformers based on the LLaMA 2 architecture (Touvron et al., 2023b): multi-head attention with $n_{\text{kv}} = n_{\text{heads}}$ (no grouped-query attention), head dimension 128, RMSNorm with $\epsilon = 10^{-5}$, and untied input and output embeddings. All models use a context length of 2048 tokens and a padded vocabulary of 50,304 (GPT-NeoX (Black et al., 2022) tokenizer, 50,277 tokens). The structural hyper-parameters for each model scale are as follows:

- **150M:** $d_{\text{model}} = 1024$, $n_{\text{layers}} = 9$, $n_{\text{heads}} = 8$, $d_{\text{ffn}} = 2688$
- **500M:** $d_{\text{model}} = 1536$, $n_{\text{layers}} = 18$, $n_{\text{heads}} = 12$, $d_{\text{ffn}} = 4096$
- **1B:** $d_{\text{model}} = 2048$, $n_{\text{layers}} = 20$, $n_{\text{heads}} = 16$, $d_{\text{ffn}} = 5440$
- **3B:** $d_{\text{model}} = 2816$, $n_{\text{layers}} = 28$, $n_{\text{heads}} = 22$, $d_{\text{ffn}} = 7488$ (≈2.9B parameters)
- **8B:** $d_{\text{model}} = 4096$, $n_{\text{layers}} = 32$, $n_{\text{heads}} = 32$, $d_{\text{ffn}} = 14336$

Pipeline partition. The L layers are split into P stages of $\lfloor L/P \rfloor$ layers each, with the remainder assigned to the last stage: 150M has 2+2+2+3 layers at P=4 and 1×7+2 at P=8; 500M 4+4+4+6 and 2×7+4; 1B 5×4 and 2×7+6; 3B 7×4; 8B 8×4. The first stage holds the token embedding and the last stage the final RMSNorm and LM head. With r=256 held fixed, the compression ratio d/r is 4×, 6×, 8×, 11× and 16× at the five scales.

F.2 Dataset and token budget

Models were pre-trained on the DCLM dataset (Li et al., 2024), pre-tokenized with the GPT-NeoX tokenizer (Black et al., 2022) into fixed-length sequences of 2048 tokens, shuffled once at the sequence level; each data-parallel rank reads a disjoint contiguous slice of the stream and reshuffles it every

epoch. For all runs, we adhered to the Chinchilla compute-optimal scaling regime (Hoffmann et al., 2022), defining our total training steps such that the models process 20 tokens per nominal parameter at $512 \times 2048 \approx 1.05$M tokens per step: the 150M model was trained for 2,862 steps (3.0B tokens), the 500M model for 9,536 steps (10.0B tokens), and the 1B model for 19,073 steps (20.0B tokens). The 3B model was also trained for 19,073 steps (20.0B tokens, ≈7 tokens per parameter, due to resource constraints). A disjoint validation shard of 20M tokens (9,764 sequences), sampled from the same shuffled stream, is held out entirely for validation; validation loss is the token-weighted cross-entropy over the whole shard.

F.3 Optimizer Configuration

We employed a hybrid optimizer approach. The 2D hidden weight matrices (attention projections and MLP matrices) were updated using the Muon optimizer (Jordan et al., 2024) with momentum 0.95 (Nesterov), PolarExpress (Amsel et al., 2026) to orthogonalize the update, the aspect-ratio scaling $\sqrt{\max(1, n_{\text{out}}/n_{\text{in}})}$, and decoupled weight decay 0.01. All remaining parameters, i.e. the token embedding, the language-modeling head, the RMSNorm gains and the anchor tables E_p^{small} were updated using AdamW (Loshchilov & Hutter, 2019) with $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$ and decoupled weight decay 0.1. The learning rate is warmed up linearly over the first 10% of the steps and then follows a cosine decay to zero at the last step; gradient clipping is off. All runs use seed 42.

Hyperparameter Sharing Strategy: To ensure a fair comparison and evaluate our approach as a direct drop-in replacement for uncompressed training under bandwidth-constrained links, we share hyperparameter configurations across all settings. Specifically, all compressed models and baselines use the exact same optimizer, learning rates, and weight decay as the uncompressed baselines, without any method-specific tuning. For the main model parameters, we set the Muon learning rate to $\eta_\mu = 0.02$ and the AdamW learning rate to $\eta_{\text{adam}} = 0.01$ (a $0.5\times$ multiplier on the Muon rate) for the 150M and 500M models. For the 1B model we used $\eta_\mu = 0.014$ and $\eta_{\text{adam}} = 0.007$, for 3B $\eta_\mu = 0.012$ and $\eta_{\text{adam}} = 0.006$, and for 8B $\eta_\mu = 0.01$ and $\eta_{\text{adam}} = 0.005$. We use a global batch size of 512 sequences (micro-batches of 4 sequences per GPU) for all token-matched runs.

F.4 MAPL Compression Hyperparameters

Our method relies on learning an orthogonal projector $A_p \in \text{St}(d, r)$ per pipeline stage boundary, initialized as the Q factor of a QR decomposition of a $d \times r$ Gaussian matrix (seed $42 + p$); the decompression matrix is tied to $A_p^\top$. We optimized these Stiefel manifold projectors using the manifold constrained steepest descent algorithm (Yang & Lai, 2026): the Riemannian gradient is formed with Muon's Nesterov momentum ($\beta = 0.95$), the descent direction is orthogonalized with PolarExpress (Amsel et al., 2026) and scaled by $\sqrt{\max(1, d/r)}$, and the ambient step is retracted onto $\text{St}(d, r)$ with a Polar Express retraction (Amsel et al., 2026) (seven degree-5 iterations in bf16). No weight decay is applied to the projectors. We found empirically that the projector optimization benefits from a reduced learning rate compared to the main model weights; therefore, we set the projector learning rate to be $0.1\times$ the main Muon learning rate (following the same warmup and cosine schedule). The factorized anchors use rank 256 at every scale, with E_p^{small} trained by AdamW and P_p frozen (Appendix A).

F.5 8B Wall-Clock Experiment

The 8B experiment of Figure 6 uses real pipeline parallelism on a single node with $4\times$ NVIDIA RTX A6000 (48 GB) GPUs, one stage per GPU, a 1F1B schedule, per-layer activation checkpointing, and NCCL point-to-point transfers. The 1 Gbps inter-stage link is emulated: after the sending GPU has finished producing a payload, the process sleeps for $\text{bytes} \times 8/\text{bandwidth}$ seconds before posting the send, so communication is fully exposed (no compute/communication overlap) and identical for every method; propagation latency is ignored and the real NCCL transfer time is measured on top of the emulated delay. Every forward transfer carries the bf16 wire tensor plus the micro-batch token ids (int32); every backward transfer carries the bf16 gradient of the wire tensor. The projector A_p is replicated on both ranks and their gradients are summed once per step with an all-reduce. The measured end-to-end traffic reduction of MAPL relative to uncompressed training is $85\times$ ($102\times$ on the activations alone; the remainder is token ids and the projector sync). Model and optimizer are the

same code as the token-matched runs: $\eta_\mu = 0.01$ ($\eta_{\text{adam}} = 0.005$, projector 0.001), Muon/AdamW weight decay 0.01/0.1, no gradient clipping, global batch 32 sequences (micro-batch 4), sequence length 2048, linear warmup over 150 steps and a cosine horizon of 1,400 steps. MAPL uses r=40 with anchor rank 256; SSN uses r=40 with Muon. Each method trains for 6 hours of training wall-clock, run one after another on the same GPUs. Within the budget, uncompressed training completes 796 optimizer steps (27.2 s/step, 3.2 GB of boundary traffic per step), MAPL 1,209 steps (17.9 s/step, 38 MB/step) and SSN 1,213 steps (17.8 s/step, 32 MB/step).

G ALGORITHM

Algorithm 2 MAPL optimizer step. Every parameter of the network is covered: the boundary projectors are updated by SPEL on $\mathrm{St}(d, r)$, the 2D hidden weights by Muon, and the remaining parameters by AdamW.

Require: Parameters partitioned into three classes: projectors $\mathcal{A} = \{A_p\}_{p=1}^{P-1} \subset \mathrm{St}(d, r)$; 2D hidden weights $\mathcal{W}$ (attention and MLP matrices of every stage); remaining parameters $\mathcal{V}$ (token embedding, LM head, anchor tables E_p^{small}, RMSNorm gains, biases).

Require: Gradients $g_\theta = \partial\mathcal{L}/\partial\theta$, accumulated over the micro-batches of one global batch, for every $\theta \in \mathcal{A} \cup \mathcal{W} \cup \mathcal{V}$.

Require: State at step t: SPEL momenta $\{m_{A_p}\}$, Muon momenta $\{M_W\}$, AdamW moments $\{(m_\theta, s_\theta)\}$; base rate η_μ; momentum β; AdamW $(\beta_1, \beta_2, \epsilon)$; weight decays λ_μ, λ_a.

$\eta_\mu \leftarrow \text{cosine_warmup}(t), \quad \alpha \leftarrow 0.1\,\eta_\mu, \quad \eta_a \leftarrow 0.5\,\eta_\mu$ ▷ *Three learning rates derived from base*

(i) Projectors — SPEL, keeps A_p on the Stiefel manifold:

for all $A_p \in \mathcal{A}$ **do**

  $g \leftarrow \partial\mathcal{L}/\partial A_p$ ▷ $\mathcal{L}$ = *Cross entropy loss*

  $g_R \leftarrow g - \frac{1}{2} A_p \left(A_p^\top g + g^\top A_p\right)$ ▷ *Projection to the tangent space*

  $m_p \leftarrow \beta\, m_p + (1-\beta)\, g_R$ ▷ *Momentum update in tangent space*

  $\Delta \leftarrow \mathrm{Orthogonalize}(m_p)$ ▷ *Muon direction from tangent momentum.*

  $A_p \leftarrow \mathrm{Orthogonalize}(A_p - 0.1\eta_\mu\, \Delta)$ ▷ *Retraction to stiefel manifold*

end for

(ii) 2D hidden weights — Muon, as in the uncompressed baseline:

for all $W \in \mathcal{W}$ of shape $n_{\text{out}} \times n_{\text{in}}$ **do**

  $M_W \leftarrow \beta\, M_W + (1-\beta)\, g_W$

  $O_W \leftarrow \sqrt{\max\left(1,\, n_{\text{out}}/n_{\text{in}}\right)}\ \mathrm{Orthogonalize}\left(\beta\, M_W + (1-\beta)\, g_W\right)$ ▷ *Nesterov momentum, then orthogonalize*

  $W \leftarrow (1 - \eta_\mu\lambda_\mu)\, W - \eta_\mu\, O_W$

end for

(iii) Embeddings, LM head, anchors, gains and biases — AdamW:

for all $\theta \in \mathcal{V}$ **do**

  $m_\theta \leftarrow \beta_1 m_\theta + (1-\beta_1)\, g_\theta, \quad s_\theta \leftarrow \beta_2 s_\theta + (1-\beta_2)\, g_\theta^2$

  $\hat{m} \leftarrow m_\theta/(1-\beta_1^t), \quad \hat{s} \leftarrow s_\theta/(1-\beta_2^t)$ ▷ *bias correction*

  $\theta \leftarrow (1 - \eta_a\lambda_a)\, \theta - \eta_a\, \hat{m}/\left(\sqrt{\hat{s}} + \epsilon\right)$

end for

The companion projectors P_p stay frozen at their random orthonormal initialisation; $\quad t \leftarrow t+1$