RESBM: RESIDUAL BOTTLENECK MODELS FOR LOW-BANDWIDTH PIPELINE PARALLELISM

Anonymous authors
Paper under double-blind review

ABSTRACT

Unlocking large-scale, low-bandwidth decentralized training has the potential to utilize otherwise untapped compute resources. In centralized settings, large-scale multi-node training is primarily enabled by data and pipeline parallelism, two techniques that typically rely on high-bandwidth communication. While efficient methods now exist for decentralized data parallelism, pipeline parallelism remains a major challenge. Recent efforts, such as Subspace Models (SM), report up to $100\times$ activation compression but rely on constrained optimization of output projection layers. We propose a different approach: the Residual Bottleneck Model (ResBM), an architecture designed from the ground up for low-bandwidth communication environments while remaining applicable to standard transformer-based architectures. ResBM introduces a residual encoder–decoder bottleneck module across pipeline boundaries that can be trained end-to-end with standard optimizers and preserves an explicit low-rank identity path. In our main comparative experiments with up to 3.5B-parameter models trained for up to 9B tokens on large-scale web datasets, we show that ResBM achieves the lowest loss among evaluated methods at extreme compression rates of $100\times$ and $128\times$. Although the authors of SM have reported little or no degradation from activation compression, our longer-horizon, optimizer-matched evaluations reveal a measurable loss gap for every compression method evaluated, including ResBM. However, ResBM consistently exhibits the smallest such gap relative to uncompressed baselines. Its loss gap remains approximately constant during later training, whereas SM's appears to grow over the evaluated horizon. We therefore position ResBM as a new state of the art for extreme activation compression in low-bandwidth pipeline-parallel training.

1 INTRODUCTION

Training frontier large language models (LLMs) requires distributing computation across hundreds or thousands of accelerators. As models grow beyond the memory capacity of individual devices, two complementary parallelism strategies are commonly combined: data parallelism (DP), which replicates the model across workers that process different data shards, and pipeline parallelism (PP), which partitions the model's layers across devices and streams activations forward and gradients backward (Narayanan et al., 2021; Huang et al., 2019). Both strategies depend on high-bandwidth interconnects, NVLink for intra-node communication and InfiniBand for inter-node transfers, making conventional large-scale training heavily dependent on specialized, expensive datacenter infrastructure

The high cost and limited availability of such clusters has motivated interest in decentralized training, where geographically distributed devices connected over the public internet collaborate to train a single model (Diskin et al., 2021; Yuan et al., 2022; Ryabinin et al., 2023). While low-communication data parallelism is now relatively mature, with methods such as DeMo (Peng et al., 2026), DiLoCo and its variants (Douillard et al., 2023; 2025; Sarfi et al., 2026) reducing DP synchronization overhead by up to $500\times$, pipeline parallelism remains the primary bottleneck. In datacenters, PP communication traverses InfiniBand links at 400 Gb/s per port (NVIDIA Corporation, 2026), whereas typical consumer upload bandwidth is around 60 Mbps (Sevilla, 2025), so naively matching datacenter transfer times would require compressing activations and their gradients by three to four orders of magnitude. However, as Ryabinin et al. (2023) note, the effective requirement is far lower than this naive ratio for larger models: by the *square-cube law*, compute grows faster than communication with model width, leaving more room to overlap the two.

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.

Compressing activations for pipeline parallelism is harder than compressing weight gradients for data parallelism: activations propagate sequentially through layers, so compression errors accumulate with depth, and naive strategies such as 8-bit quantization or autoencoder-like bottlenecks between blocks either save little or degrade convergence (Section 2). This suggests that where compression is applied matters as much as its magnitude: in particular, placing compression on the residual identity path, the connection that makes deep networks trainable (He et al., 2016), risks destabilizing optimization.

Recently, Subspace Models (SM) were proposed by Ramasinghe et al. (2025), claiming to achieve up to $100\times$ activation compression without convergence degradation. Their approach constrains embedding and out-projection matrices in each transformer layer to a shared low-rank subspace: out-projection matrices remain trainable but are projected into this subspace during optimization, while the embedding is decomposed into trainable low-rank and fixed high-rank components, allowing low-rank activations to be transmitted and reconstructed at the receiving node. This requires constrained optimization, with the shared subspace periodically updated on the Grassmann manifold, departing from standard end-to-end training and restricting the effective parameter space available to the model. Moreover, the reported absence of degradation was established over a relatively short training horizon. Our extended experiments over a larger token budget suggest that the cost of these constraints becomes visible later in training, such that short training horizons can understate their effect (Section 4.2).

In this paper, we take a different approach. Rather than retrofitting compression into architectures designed for centralized training, we propose the Residual Bottleneck Model (ResBM), an architecture designed specifically for low-bandwidth pipeline parallelism. ResBM introduces learnable subspace projection layers that reduce the dimensionality of communicated activations while preserving an identity path through the bottleneck. Because the bottleneck is integrated into the residual architecture rather than applied as an external compression mechanism, these projection layers can be trained end-to-end as ordinary model parameters using off-the-shelf optimizers. Informed by longer-horizon training ablations, we relax the lossless-compression assumption and instead study how much degradation extreme activation compression induces over extended pretraining horizons and across model sizes, optimizers, compression rates, and numbers of pipeline stages. Across these settings, ResBM consistently exhibits the smallest degradation among the extreme activation-compression methods we evaluate, establishing it as the new state of the art for extreme activation compression in low-bandwidth pipeline-parallel training.

2 Related Work

Decentralized training and activation compression. Decentralized training distributes computation across autonomous, geographically dispersed devices connected by heterogeneous networks (Diskin et al., 2021). Systems such as SWARM parallelism (Ryabinin et al., 2023) and Tasklets (Yuan et al., 2022) address scheduling and resource allocation in such environments. SWARM additionally explores compression-aware architectures using 8-bit quantization or bottleneck layers applied to the full block output, including the residual stream, but only at modest 2–$4\times$ compression. We use this bottleneck as an ablation baseline, where adding ResBM's residual identity path isolates its architectural contribution (Section 4.2).

Activation compression for pipeline parallelism is particularly challenging because lossy errors propagate across layers and can accumulate with model depth (Bian et al., 2024), while activation distributions also vary substantially across samples and batches (Rudakov et al., 2023). Standard approaches such as top-k sparsification, SVD projection, and quantization therefore degrade or fail at aggressive compression rates (Ramasinghe et al., 2025). Subspace Models (SM) address this by constraining embedding and out-projection matrices to a shared low-rank subspace, claiming up to $100\times$ compression with minimal degradation. This constraint restricts the model's effective parameter space. The original SM formulation additionally adapts the shared subspace through periodic Grassmann-manifold optimization, introducing a separate optimization procedure. Using the authors' released SM implementation from Agora (Avraham et al., 2026), we find that over longer training horizons SM develops a loss gap relative to the uncompressed baseline that grows with tokens seen. ResBM instead uses learned residual bottleneck modules trained end-to-end with standard optimizers and exhibits superior convergence properties (Section 4.2).

Residual connections and identity preservation. The residual connection, introduced by He et al. (2016), is a cornerstone of modern deep network design. By adding an identity shortcut around each stack of layers, it addresses the degradation problem of very deep plain networks, making them easier to optimize and enabling training of substantially deeper architectures. In transformer-based LLMs, the residual stream carries information across all layers and is essential for stable optimization. Recent work by Xie et al. (2026) has shown that some transformations can be safely introduced along the residual path provided they preserve the identity mapping property of the connection. Our architecture builds on this insight: the bottleneck projection operates alongside the residual stream rather than on it, ensuring that a low-rank identity path remains uninterrupted even as the communicated activation dimensionality is reduced.

3 Residual Bottleneck Models

3.1 Notation

Modern LLMs are typically implemented as a stack of Transformer blocks (Vaswani et al., 2017), each built around self-attention and the residual connection. A single residual sublayer of such a block can be written as

$$x_{i+1} = x_i + F(x_i, W_i), \tag{1}$$

where $x_i \in \mathbb{R}^{L\times H}$ denotes the input to residual sublayer i, with L the sequence length and H the hidden dimension. The residual function $F(\cdot, W_i)$ typically corresponds to either a self-attention or a feed-forward sublayer parameterized by weights W_i; each such sublayer performs one residual addition and therefore advances the index by one, so a Transformer block occupies two consecutive sublayer indices. Activations are represented with one token per row, so a linear layer with weight matrix W acts by right multiplication, $y = xW$, and its outputs lie in the row space of W.

More generally, if the hidden dimensionality is allowed to change from one sublayer to the next, the residual connection must account for this mismatch. Let N denote the number of residual sublayers and let c_i denote the width of the residual stream entering sublayer i, so that $x_i \in \mathbb{R}^{L\times c_i}$ and $F(x_i, W_i) \in \mathbb{R}^{L\times c_{i+1}}$.

The identity branch must be mapped to the width of the sublayer output before the two can be added. We do this with a fixed, parameter-free projection built from rectangular identity matrices. For integers $p, q \geq 1$ and $0 \leq k \leq \min(p, q)$, let $\mathbf{I}^{(k)}_{p\times q} \in \mathbb{R}^{p\times q}$ denote the rank-k rectangular identity matrix defined elementwise as

$$\left[\mathbf{I}^{(k)}_{p\times q}\right]_{a,b} = \begin{cases} 1, & \text{if } a = b \text{ and } a \leq k, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

and write $\mathbf{I}_{p\times q} := \mathbf{I}^{(\min(p,q))}_{p\times q}$ for the full-rank case, so that $\mathbf{I}_{p\times p}$ is the usual identity matrix $\mathbf{I}_p$. The *identity projection operator* of sublayer i is the matrix

$$\mathcal{P}^{\mathrm{id}}_i := \mathbf{I}_{c_i\times c_{i+1}} \in \mathbb{R}^{c_i\times c_{i+1}}, \qquad \text{so that} \qquad x_i\mathcal{P}^{\mathrm{id}}_i \in \mathbb{R}^{L\times c_{i+1}} \quad \text{for} \quad x_i \in \mathbb{R}^{L\times c_i}, \tag{3}$$

acting on the identity branch by right-multiplication. If $c_i < c_{i+1}$ it zero-pads the identity branch, appending $c_{i+1} - c_i$ zero columns; if $c_i > c_{i+1}$ it truncates the last $c_i - c_{i+1}$ columns; and if $c_i = c_{i+1}$ it reduces to the standard identity matrix $\mathbf{I}_{c_i}$. The standard Transformer is recovered by setting $c_i = H$ for all i. The generalized residual update becomes

$$x_{i+1} = x_i\mathcal{P}^{\mathrm{id}}_i + F(x_i, W_i), \tag{4}$$

and unrolling it from any sublayer m to any later sublayer n, with $0 \leq m < n \leq N$, gives

$$x_n = x_m \left(\prod_{i=m}^{n-1} \mathcal{P}^{\mathrm{id}}_i\right) + \sum_{i=m}^{n-1} F(x_i, W_i) \left(\prod_{j=i+1}^{n-1} \mathcal{P}^{\mathrm{id}}_j\right), \tag{5}$$

where products of matrices are taken in order of increasing index from left to right, and an empty product denotes the identity matrix. Section 3.3 shows that these projections preserve the identity mapping property of the residual connection.

3.2 THE BOTTLENECK LAYER

The bottleneck layer is introduced after the feed-forward layer of every Transformer block except the last, whose output is consumed locally and never crosses a pipeline boundary. It consists of an encoder, which maps the feed-forward output to a lower-dimensional bottleneck space before the boundary, and a decoder, which maps the received state back to the working width after it; each is a two-layer bias-free MLP with an elementwise nonlinearity after each linear layer.

Setting. Consider a pipeline boundary and let i be the index, in the sublayer indexing of Section 3.1, of the feed-forward sublayer of the block before the boundary, so that $i+1$ is the attention sublayer of the block after it. Writing h for the bottleneck width, the widths are $c_i = H$, $c_{i+1} = h$ and $c_{i+2} = H$, so the widths of a ResBM alternate between H and h, and by Equation 3 the identity projections at the two sublayers are

$$\mathcal{P}_i^{\mathrm{id}} = \mathbf{I}_{H\times h}, \qquad \mathcal{P}_{i+1}^{\mathrm{id}} = \mathbf{I}_{h\times H},$$

the first truncating the identity branch to its leading h coordinates and the second zero-padding it back to width H. The only sublayers not of these two kinds are the attention sublayer of the first block, which receives the H-wide embedding, and the feed-forward sublayer of the last block, whose output is not compressed; both have equal input and output widths and the standard identity $\mathbf{I}_H$. Attn_i and FF_i denote the complete attention and feed-forward sublayer maps, including their post-normalization, and W_i^{Attn} and W_i^{FF} their parameters.

Encoder and transmitted state. Let $x_i \in \mathbb{R}^{L\times H}$ be the post-attention state entering the feed-forward sublayer i, and $z_i = \mathrm{FF}_i(x_i) \in \mathbb{R}^{L\times H}$ the feed-forward output, immediately before compression. The encoder E_i, with parameters W_i^E, maps it to the *compressed residual contribution*

$$b_i = E_i(z_i), \qquad b_i \in \mathbb{R}^{L\times h}, \qquad h < H.$$

Concretely, writing ϕ for the elementwise non-linearity and m for the intermediate width of the encoder and decoder,

$$E_i(z_i) = \phi\big(\phi\big(z_i\, W_{i,1}^E\big)\, W_{i,2}^E\big), \qquad W_{i,1}^E \in \mathbb{R}^{H\times m}, \quad W_{i,2}^E \in \mathbb{R}^{m\times h}, \qquad m = H/4, \tag{6}$$

so that $W_i^E = \{W_{i,1}^E, W_{i,2}^E\}$ and the encoder maps $H \to m \to h$.

Because E_i is part of the residual function of sublayer i, its output is summed with the projected identity branch before transmission, and the tensor sent across the boundary is the *compressed residual state*

$$x_{i+1} = x_i \mathcal{P}_i^{\mathrm{id}} + b_i, \qquad x_{i+1} \in \mathbb{R}^{L\times h}, \tag{7}$$

which is Equation 4 at sublayer i with residual function $F(\cdot, \{W_i^{\mathrm{FF}}, W_i^E\}) = E_i \circ \mathrm{FF}_i$; the encoder parameters belong to the residual function but not to FF_i, so the parameter set of the sublayer is the union of the two.

Decoder and expanded state. On the receiving stage, the decoder D_{i+1}, with parameters W_{i+1}^D, acts on the *received state* x_{i+1}:

$$\tilde{x}_{i+1} = D_{i+1}(x_{i+1}), \qquad \tilde{x}_{i+1} \in \mathbb{R}^{L\times H}, \tag{8}$$

with the decoder being the mirror image of the encoder:

$$D_{i+1}(x_{i+1}) = \phi\big(\phi\big(x_{i+1}\, W_{i+1,1}^D\big)\, W_{i+1,2}^D\big), \qquad W_{i+1,1}^D \in \mathbb{R}^{h\times m}, \quad W_{i+1,2}^D \in \mathbb{R}^{m\times H}, \tag{9}$$

so that $W_{i+1}^D = \{W_{i+1,1}^D, W_{i+1,2}^D\}$ and the decoder maps $h \to m \to H$. The decoder output $\tilde{x}_{i+1}$ is the tensor that enters the attention map of sublayer $i+1$. Because x_{i+1} already carries the projected identity branch, $\tilde{x}_{i+1}$ is an expansion of the compressed residual state rather than a reconstruction of z_i: despite its encoder–decoder structure, the bottleneck is not an autoencoder and is not trained under a reconstruction objective. Equation 4 at sublayer $i+1$, with residual function $F(\cdot, \{W_{i+1}^{\mathrm{Attn}}, W_{i+1}^D\}) = \mathrm{Attn}_{i+1} \circ D_{i+1}$, restores the working width:

$$x_{i+2} = x_{i+1}\mathcal{P}_{i+1}^{\mathrm{id}} + \mathrm{Attn}_{i+1}(\tilde{x}_{i+1}), \qquad x_{i+2} \in \mathbb{R}^{L\times H}. \tag{10}$$

The state x_{i+2} enters the feed-forward sublayer $i+2$ of the receiving block; if that block also precedes a boundary, Equation 7 applies to it in turn.

Figure A.1 shows the resulting layout: the encoder ends the sending block, the decoder starts the receiving one, communication scales with h rather than H, and the identity branch bypasses both, acted on only by the fixed projections $\mathcal{P}_i^{\mathrm{id}}$ and $\mathcal{P}_{i+1}^{\mathrm{id}}$, which appear in the figure as splits and concatenations of the residual stream.

3.3 The identity path

Building on the rank-collapse motivation of Ramasinghe et al. (2025) and the output-projection spectra in Section B.1, we investigate whether the identity path can likewise be narrowed. If its forward activations and backward gradients are sufficiently concentrated, this may permit lower-dimensional communication with limited impact on optimization. To retain a direct shortcut through repeated width changes, we choose the fixed projections $\mathcal{P}_i^{\mathrm{id}}$ of Equation 3 to preserve an exact identity mapping on the communicated subspace.

Substituting Equation 3 into Equation 5, every product of identity projections telescopes into a single projection: for any sequence of widths $c_0, c_1, \ldots, c_N$ and any $0 \le m < n \le N$,

$$\prod_{i=m}^{n-1} \mathbf{I}_{c_i \times c_{i+1}} = \mathbf{I}^{(k)}_{c_m \times c_n}, \qquad k = \min_{m \le i \le n} c_i. \tag{11}$$

The composition is the $c_m \times c_n$ rectangular identity truncated to rank k, the narrowest width anywhere along the path: a coordinate survives the whole composition precisely when it is present at every intermediate width. For the alternating widths of Section 3.2, measured from one communication boundary to the next, the composition is

$$\mathbf{I}_{h \times H}\, \mathbf{I}_{H \times h} = \mathbf{I}_h,$$

a full-rank identity on the h-dimensional communicated stream: that stream carries an exact, unbroken identity path from one pipeline stage to the next, of any length. Measured in the H-dimensional working width, from one mid-block state to the next, the same block contributes

$$\mathbf{I}_{H \times h}\, \mathbf{I}_{h \times H} = \mathbf{I}^{(h)}_{H \times H},$$

so any path crossing at least one communication boundary has $k = h$, independently of its length.

This construction preserves an exact identity shortcut on the communicated h-dimensional subspace, providing a direct gradient contribution that bypasses the learned encoder and decoder transformations. These coordinates have no predefined semantic interpretation; end-to-end training can adapt the surrounding representations to route the most expressed directions through the preserved shortcut.

4 Experiments

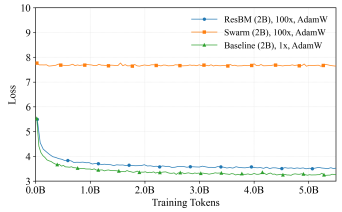
4.1 Experimental setup

Unless otherwise stated, all experiments use a baseline transformer model with approximately 2B parameters, based on the OLMo-2 architecture (OLMo et al., 2025) with reordered (RMS) normalization (Liu et al., 2022) and QK-Norm (Dehghani et al., 2023). We use SiLU as the nonlinearity ϕ in both the encoder and decoder. The default 2B model has 8 transformer blocks, hidden width $H = 4096$, a 50K-token vocabulary, and a 1024-token context. We train on the C4 dataset (Raffel et al., 2020) with a batch size of 32. The default optimizer is AdamW (Loshchilov & Hutter, 2019) with a peak learning rate of 3×10^{-4}, 500 warmup steps, and a weight decay of 0.01.

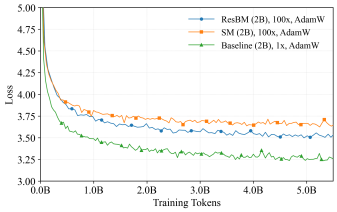
Unless otherwise specified, all compression methods we evaluate build upon the baseline architecture by introducing a compression boundary after each transformer block except the last one, resulting in seven bottleneck layers in total. These layers reduce the dimensionality of communicated activations and their corresponding gradients by a factor equal to the specified compression rate.

4.2 Performance under extreme compression

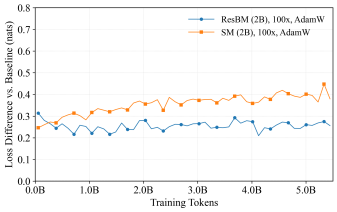
Scaling compression ratios. We first compare ResBM against SM (Ramasinghe et al., 2025) and SWARM (Ryabinin et al., 2023) under extreme compression ratios of $102.4\times$ (denoted $100\times$)



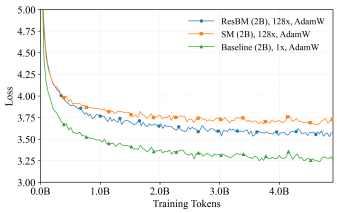
(a) $\text{Loss}_{100\times}$ (ResBM vs. SWARM).



(b) $\text{Loss}_{100\times}$ (ResBM vs. SM).



(d) Evolution of loss gap (100×).



(c) $\text{Loss}_{128\times}$ (ResBM vs. SM).



(e) Evolution of loss gap (128×).

Figure 1: **Training loss under extreme compression (AdamW).** Comparison of ResBM, SM, and SWARM on 2B-parameter models trained with AdamW at $100\times$ and $128\times$ compression. Right panels show the evolution of the loss gap relative to the corresponding uncompressed baseline.

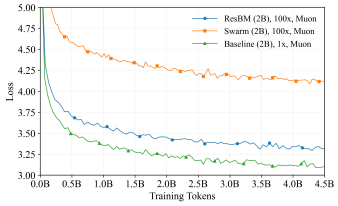
and $128\times$, corresponding to bottleneck dimensions of 40 and 32, respectively. We use the authors' Agora implementation of SM (Avraham et al., 2026), which enforces the output-projection subspace constraints through reparameterization, allowing the trainable parameters to be optimized with off-the-shelf optimizers such as AdamW and Muon. We compare the loss curves of the three architectures against an uncompressed baseline trained with the same optimizer (AdamW). Each run covers around 5B tokens under the configuration of Section 4.1. Figure 1 shows the loss curves of the three architectures. ResBM outperforms SM at both compression rates. The loss-gap plots show that ResBM's gap relative to the uncompressed baseline remains approximately stable, while SM's grows over training. SWARM, however, fails to train effectively under this extreme regime. As a sanity check, Appendix Figure C.1 reports SWARM's loss with 1, 2, 3, and 7 bottlenecks, showing progressively improved training behavior as the number of bottlenecks is reduced.

Optimizing with Muon. To evaluate the compatibility of ResBM with different optimizers, we choose Muon as an alternative to the default AdamW optimizer. We use a Nesterov momentum of 0.95 and a learning rate of 5.0×10^{-3}, and evaluate two compression ratios, $100\times$ and $128\times$. As shown in Figure 2, ResBM reaches a lower loss than both SWARM and SM under Muon optimization. More importantly, switching from AdamW to Muon reduces ResBM's loss gap to the uncompressed baseline by roughly 0.05 nats at both compression rates, whereas SM remains at approximately the same loss gap.

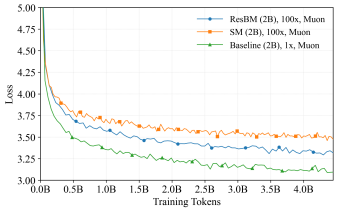
We hypothesize that this difference is related to how the two methods use the available representation space. SM constrains part of the model to a shared low-rank subspace, whereas ResBM leaves the main representation unconstrained and learns a dedicated encoder from the full activation stream. Since Muon exhibits less rank collapse than AdamW (Figure B.1), this may benefit ResBM more directly by allowing its bottleneck to exploit a richer representation before compression.

Scaling model size. We further evaluate ResBM in a more extreme scaling setting by compressing a 3.5B-parameter model with 16 transformer blocks, resulting in 15 bottleneck layers, one after each block except the last. We train the model for 9B tokens and compare ResBM against SM. As shown in Figure 3, ResBM again achieves a lower loss than SM. Moreover, ResBM's loss gap relative to the uncompressed baseline remains approximately stable, or grows only slightly, over training, whereas the gap for SM continues to increase with the number of tokens seen.

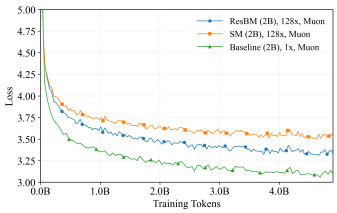
Varying context length and dataset. To evaluate ResBM's robustness in a longer-context setting on a different dataset, we extend the context window to 2048 tokens and switch to FineWeb-



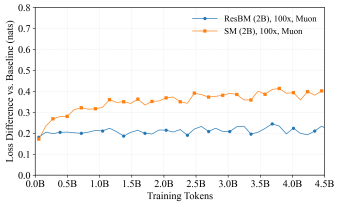
(a) $\text{Loss}_{100\times}$ (ResBM vs. SWARM).



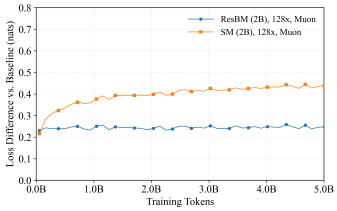
(b) $\text{Loss}_{100\times}$ (ResBM vs. SM).



(c) $\text{Loss}_{128\times}$ (ResBM vs. SM).

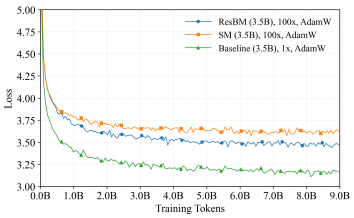


(d) Evolution of loss gap (100×).

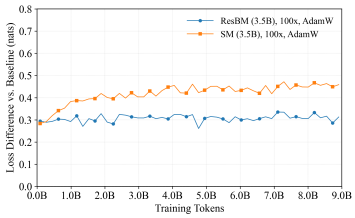


(e) Evolution of loss gap (128×).

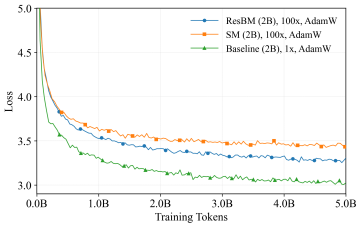
Figure 2: **Training loss under extreme compression (Muon).** Comparison of ResBM, SM, and SWARM on 2B-parameter models trained with Muon at 100× and 128× compression. Right panels show the evolution of the loss gap relative to the corresponding uncompressed baseline.



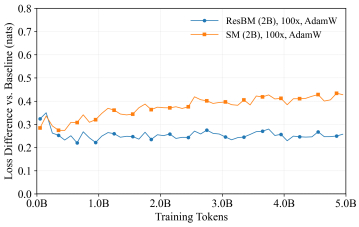
(a) $\text{Loss}_{3.5B}$ (ResBM vs. SM).



(b) Evolution of loss gap (3.5B).



(c) $\text{Loss}_{L=2048}$ (ResBM vs. SM).

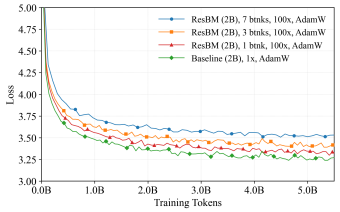


(d) Evolution of loss gap ($L = 2048$).

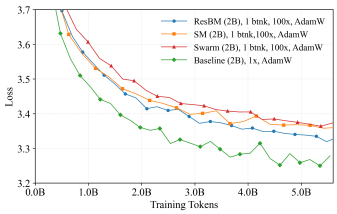
Figure 3: **Loss under model-size and context-length scaling.** (a–b) 100× compression on a 3.5B model with 16 transformer blocks. (c–d) The 2B model with context length 2048 on FineWeb-Edu2. Right panels show the loss gap to the corresponding uncompressed baseline.

Edu2 (`HuggingFaceFW/fineweb-edu-score-2`), which contains curated educational content (Penedo et al., 2024). As shown in Figure 3, ResBM exhibits the same trend observed in the previous experiments, outperforming SM while maintaining a loss-gap behavior consistent with the earlier settings. In particular, ResBM's gap relative to the uncompressed baseline remains approximately stable over training, whereas the gap for SM shows a stronger tendency to grow with the number of tokens seen.

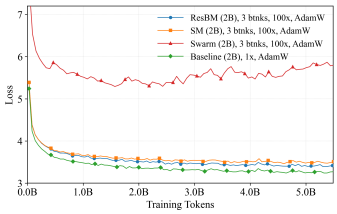
Varying the number of bottleneck layers. To test whether ResBM's performance advantage is specific to the extreme setting in which compression is applied after every transformer block, we keep the compression rate fixed at 100× while reducing the number of bottlenecks. We consider a single bottleneck placed after the fourth transformer block, as well as three bottlenecks inserted after every other block. ResBM retains the best performance under both settings, and its loss gap relative to the



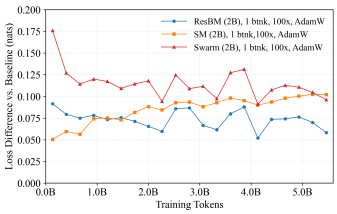
(a) ResBM with 1, 3, and 7 btnks.



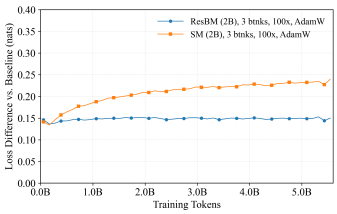
(b) $\text{Loss}_{\text{1-btnk}}$ (ResBM vs. all).



(c) $\text{Loss}_{\text{3-btnks}}$ (ResBM vs. all).



(d) Evolution of loss gap (1 btnk).



(e) Evolution of loss gap (3 btnks).

Figure 4: **Loss under varying numbers of bottlenecks.** 2B models trained with AdamW at 100× compression. Left: comparison of ResBM with 1, 3, and 7 bottlenecks. (b,d) ResBM versus the other compression methods with 1 bottleneck. (c,e) ResBM versus the other compression methods with 3 bottlenecks. Here, "btnk" denotes a bottleneck layer.

Table 1: Training throughput (tokens per second) for the 2B model under pipeline parallelism across 8 L40S GPUs. The centralized baseline runs over a 10 Gbps link; all other rows run over 80 Mbps.

		TPS	
Configuration	**Compression**	**AdamW**	**Muon**
Centralized baseline (10 Gbps)	–	18,084	16,478
Decentralized baseline	–	605	604
ResBM	100×	18,053	16,378
ResBM	128×	18,058	16,382

uncompressed baseline exhibits dynamics consistent with those observed in the previous experiments as shown in Figure 4.

4.3 Throughput Analysis

The convergence experiments above ran on a single GPU and involved no communication; we now measure end-to-end training throughput under pipeline parallelism. All throughput experiments in this section use 8× NVIDIA L40S GPUs with GPipe-style pipeline parallelism implemented via PyTorch distributed over NCCL. Each transformer block is assigned to a separate GPU, with inter-stage activations and gradients transmitted over the public internet in the decentralized configurations. We compare four uncompressed 2B baselines, trained with either AdamW or Muon over a 10 Gbps datacenter link (**centralized**) or an 80 Mbps internet link (**decentralized**), against four ResBM variants at 80 Mbps, crossing the same two optimizers with two compression ratios (100×, 128×). The 80 Mbps setting matches the decentralized bandwidth used by Ramasinghe et al. (2025). All eight configurations use the 2B-parameter architecture and hyperparameters of Section 4.1.

Table 1 reports throughput in tokens per second for each configuration, averaged over the full 4 hours of training. Under both optimizers, the uncompressed decentralized baseline suffers a dramatic throughput collapse at 80 Mbps, while ResBM recovers nearly all of the centralized throughput despite operating over a 125× slower link. Appendix D reports throughput gains across bandwidth settings and the parameter overhead of ResBM.

4.4 PRACTICAL CONSIDERATIONS.

Taken together, our experiments show that ResBM consistently outperforms previous compression methods across different regimes. Beyond this overall advantage, three practical observations emerge.

First, after the initial phase of training, ResBM exhibits an approximately constant loss offset relative to the uncompressed baseline. This suggests ResBM may preserve the token-scaling behavior of the underlying model, which could make existing scaling laws useful for predicting compressed training performance. Confirming this requires experiments over larger token budgets and model scales.

Second, our main experiments deliberately consider an extreme setting in which compression is applied after almost every transformer block. In practice, the required number of bottlenecks depends on the size and memory capacity of the participating nodes. As shown in Figure 4, reducing the number of bottlenecks preserves ResBM's advantage while bringing its loss much closer to the uncompressed baseline, suggesting that practical deployments may incur substantially smaller degradation than our most extreme settings.

Finally, Muon provides a particularly useful practical configuration for ResBM. At $100\times$ and $128\times$ compression, ResBM trained with Muon reaches the loss of the uncompressed AdamW baseline after approximately 17B and 26B tokens, respectively (Figure C.2). Thus, extreme communication compression can still recover the performance of conventional centralized AdamW training. A gap remains relative to the uncompressed Muon baseline, however, leaving room for further work on closing the optimizer-matched gap.

5 CONCLUSION

We introduced the Residual Bottleneck Model (ResBM), an architecture designed natively for low-bandwidth pipeline-parallel training. Unlike prior compression methods, ResBM preserves an explicit identity pathway while learning its residual bottlenecks end-to-end with off-the-shelf optimizers, adding small parameter and compute overhead while remaining effective under extreme compression rates of $100\times$ and $128\times$. Our extended pretraining runs show that extreme activation compression incurs a training-loss penalty for every method we evaluate, including our own. ResBM exhibits the smallest late-training gap, which remains approximately stable after the initial phase, whereas SM's gap grows over the evaluated horizon. In realistic decentralized settings over internet-grade links, ResBM recovers near-centralized throughput under severe bandwidth constraints. We therefore establish ResBM as a new state of the art for extreme activation compression in low-bandwidth pipeline-parallel training.

Broader impact. By achieving the smallest training-loss penalty among the evaluated methods at $100\times$ and $128\times$ compression, ResBM further reduces the communication barrier to pipeline-parallel pretraining over low-bandwidth links, making decentralized pretraining of large models more practical. This is a step toward more accessible and, potentially, more environmentally sustainable training, by enabling better utilization of geographically distributed idle compute instead of concentrating capacity in a small number of hyperscale datacenters.[1]

Limitations. Our experiments cover 2B and 3.5B parameter models, main comparisons up to 9B tokens with supplementary 2B runs up to 26B tokens, C4 and FineWeb-Edu2, context lengths of 1024 and 2048, AdamW and Muon, compression rates of $100\times$ and $128\times$, and 1, 3, 7, and 15 bottlenecks. Nevertheless, frontier-scale models, substantially longer training, additional architecture families, and downstream tasks remain unexplored. Across the evaluated settings, ResBM's loss gap is front-loaded and approximately stable, whereas SM's grows during training. These dynamics require validation at larger model and token scales. Further exploration of the bottleneck design space is also left to future work.

[1] ResBM already powers a real-world decentralized pretraining project, demonstrating its applicability beyond controlled experimental settings. The code to reproduce the proposed architecture is open source and available as part of that project's codebase; the link is omitted here to preserve anonymity and will be provided in the camera-ready version.

AI USE STATEMENT

In this work, we have not used generative AI tools for any of the tasks with required disclosure, including developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, writing proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, or interpreting results. The remaining tasks with required disclosure, including generating synthetic datasets, cleaning and reformatting datasets, assisting with translation, and supporting qualitative or thematic data analysis, are not applicable to this work. Additionally, we used generative AI tools to edit the research paper to improve readability and grammar. We have reviewed all AI-assisted work, and all authors have read the paper in its entirety several times. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Gil Avraham, Violetta Shevchenko, Karol Pajak, James Snewin, Harry Xi, Hadi Mohaghegh Dolatabadi, Thalaiyasingam Ajanthan, Sameera Ramasinghe, Chamin Hewa Koneputugodage, Shamane Siriwardhana, and Alexander Long. Pluralis 8b: Asynchronous large-scale distributed training over the internet, 2026. URL https://github.com/PluralisResearch/agora.

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

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd Van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. 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. 7480–7512. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, quentin lhoest, Anton Sinitsin, Dmitry Popov, Dmitry V. Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, Denis Mazur, Ilia Kobelev, Yacine Jernite, Thomas Wolf, and Gennady Pekhimenko. Distributed deep learning in open collaborations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 7879–7897. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/41a60377ba920919939d83326ebee5a1-Paper.pdf.

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

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training, 2025. URL https://arxiv.org/abs/2502.16982.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11999–12009, 2022. doi: 10.1109/CVPR52688.2022.01170.

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. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

NVIDIA Corporation. NVIDIA Quantum-2 InfiniBand platform. Product page, 2026. URL https://www.nvidia.com/en-us/networking/quantum2/. Accessed 2026-09-19.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. 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. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Bowen Peng, Lizhang Chen, Baiyu Su, Jeffrey Quesnelle, Diederik P Kingma, and Qiang Liu. DeMo: Decoupled Momentum Optimization. In *International Conference on Learning Representations*, 2026. URL https://mlanthology.org/iclr/2026/peng2026iclr-demo/.

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. URL http://jmlr.org/papers/v21/20-074.html.

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. Subspace Networks: Scaling Decentralized Training with Communication-Efficient Model Parallelism. In *Advances in Neural Information Processing Systems*, 2025. URL https://mlanthology.org/neurips/2025/ramasinghe2025neurips-subspace/.

Mikhail I. Rudakov, Aleksandr Nikolaevich Beznosikov, Ya A. Kholodov, and Alexander Vladimirovich Gasnikov. Activations and gradients compression for model-parallel training. In *Doklady Mathematics*, volume 108, pp. S272–S281. Springer, 2023.

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. SWARM parallelism: Training large models can be surprisingly communication-efficient. 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. 29416–29440. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/ryabinin23a.html.

Amir Sarfi, Benjamin Thérien, Joel Lidin, and Eugene Belilovsky. Communication efficient LLM pre-training with sparseloco, 2026. URL https://openreview.net/forum?id=gBppFpDjyW.

Jaime Sevilla. How far can decentralized training over the internet scale?, 2025. URL https://epoch.ai/gradient-updates/how-far-can-decentralized-training-over-the-internet-scale. Accessed: 2026-09-19.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mhc: Manifold-constrained hyper-connections, 2026. URL https://arxiv.org/abs/2512.24880.

Binhang Yuan, Yongjun He, Jared Quincy Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Ré, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

A RESBM ARCHITECTURE

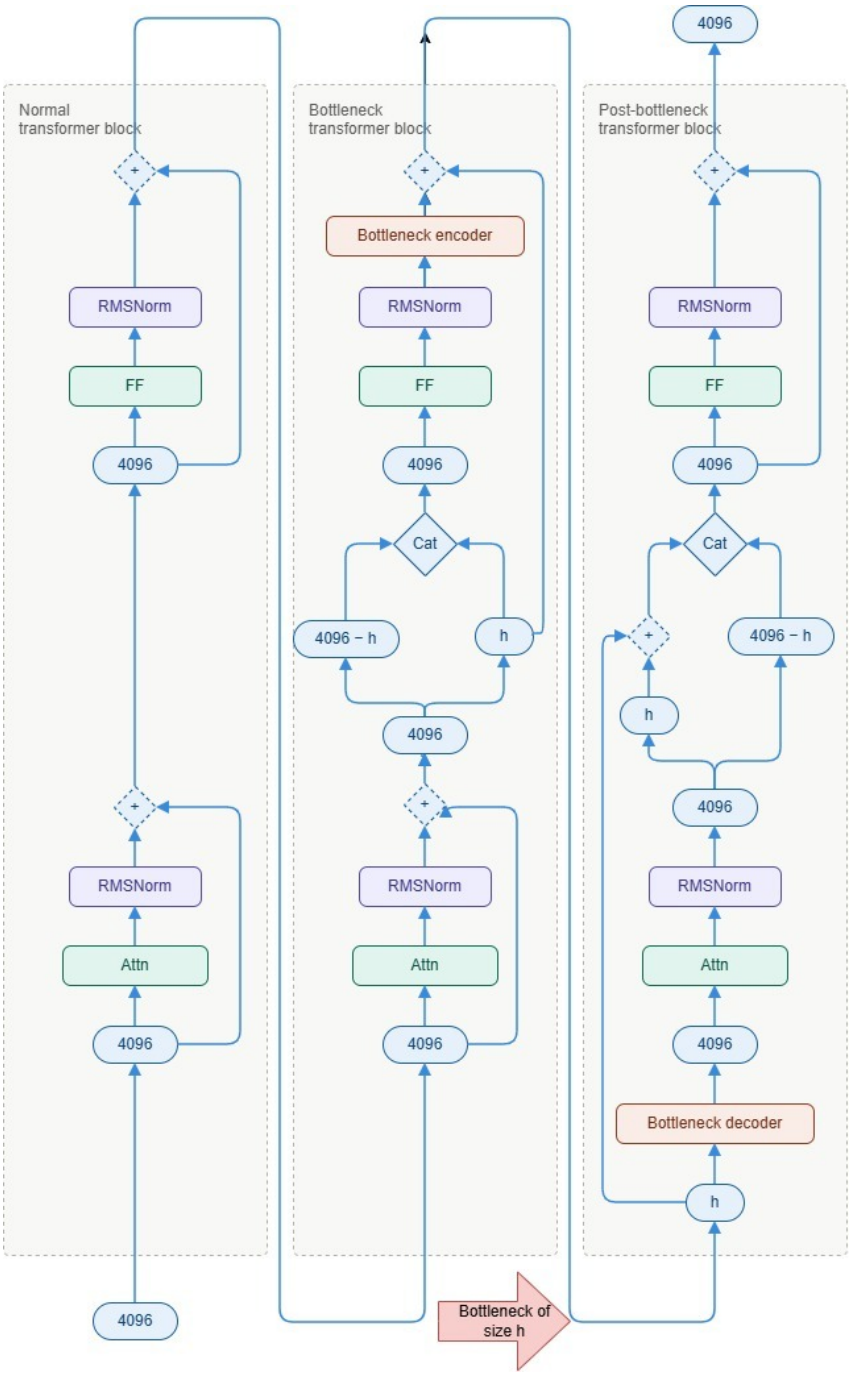


Figure A.1: **ResBM architecture across a pipeline-parallel communication boundary.** For clarity, we show only three transformer blocks and a single encoder–decoder bottleneck instance. The encoder compresses the activation to bottleneck size h before communication, and the decoder expands it back to the full-width representation H at the receiving stage.

B SPECTRAL ANALYSIS

B.1 BASELINE OUTPUT PROJECTION SPECTRA

Figure B.1 shows the singular value spectra of the FFN output projection matrices of the uncompressed 2B baseline trained with AdamW and with Muon, computed following the protocol of Liu et al. (2025).

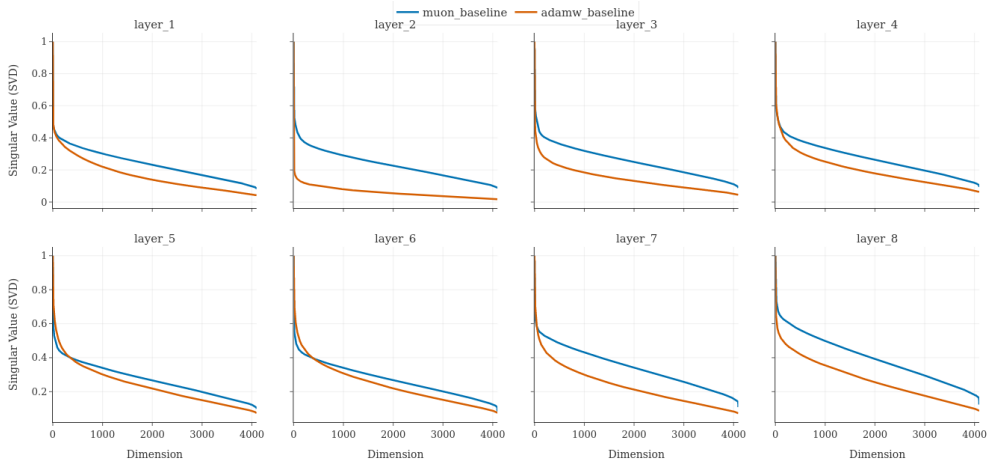


Figure B.1: **Singular value analysis of the FFN output projection matrices of the uncompressed baseline.** Both AdamW and Muon exhibit substantial singular value decay, indicating a significant degree of effective rank collapse. However, AdamW shows a markedly faster decay, whereas Muon maintains larger singular values across more dimensions, suggesting weaker effective rank collapse than AdamW.

C ADDITIONAL CONVERGENCE EXPERIMENTS

C.1 SWARM LOSS UNDER DIFFERENT BOTTLENECK CONFIGURATIONS

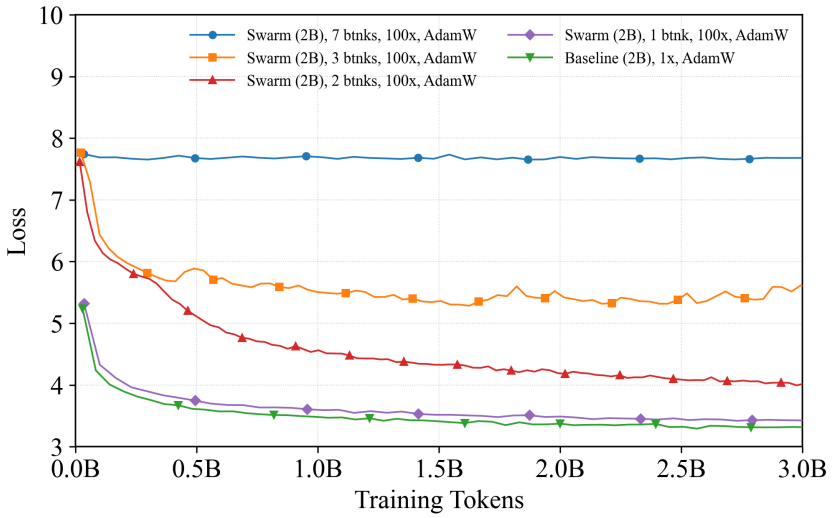
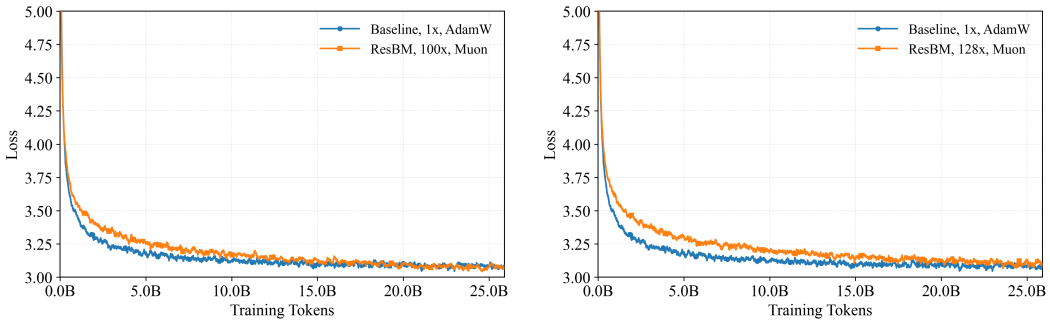


Figure C.1: **SWARM convergence under varying bottleneck counts.** Training loss for SWARM with 1, 2, 3, and 7 bottlenecks, showing that convergence degrades significantly as the number of bottlenecks increases.

C.2 LOSS OF RESBM WITH MUON COMPARED TO ADAMW BASELINE



(a) 100× ResBM (Muon) vs. Baseline.

(b) 128× ResBM (Muon) vs. Baseline.

Figure C.2: **Practical comparison of Muon-trained ResBM against the uncompressed AdamW baseline.** We compare ResBM trained with Muon at 100× and 128× compression against the corresponding uncompressed AdamW baseline. Despite operating under extreme communication compression, ResBM reaches the AdamW baseline loss after sufficient training at both compression rates. This provides a pragmatic reference point: although an optimizer-matched gap remains relative to an uncompressed Muon model, ResBM with Muon can recover the performance attained by conventional centralized training with AdamW while substantially reducing communication.

D ADDITIONAL SYSTEMS EXPERIMENTS

D.1 WALL-CLOCK CONVERGENCE

The experiments in this subsection and in Appendix D.2 were run on 8× NVIDIA A10G GPUs rather than the L40S GPUs of Section 4.3, with the same pipeline-parallel configuration otherwise; absolute throughputs therefore differ from Table 1. Beyond raw throughput, we compare training loss as a function of wall-clock time to assess whether the compression-induced throughput gains translate into faster convergence in practice. Figure D.1 plots the training loss against elapsed time for all eight configurations of Section 4.3: the centralized and decentralized uncompressed baselines and the 100× and 128× ResBM variants, each under AdamW (left) and Muon (right). Each curve shows the loss over the first 35M tokens. With a bandwidth of 80 Mbps, both ResBM variants closely track the wall-clock convergence of the centralized baseline, reaching 35M tokens within a few minutes of it, consistent with the near-centralized throughput of Table 1. The lower throughput of the Muon variants relative to the AdamW variants is attributable to Muon's higher per-step compute cost, which does not affect communication volume. The uncompressed decentralized baseline converges significantly slower as the pipeline stall induced by transmitting full-rank activations over the restricted link reduces throughput.

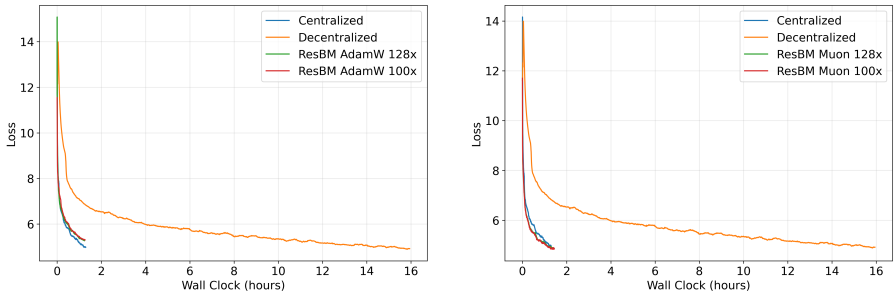


Figure D.1: Training loss versus wall-clock time on C4 over the first 35M tokens for all runs. ResBM at 80 Mbps closely matches the centralized baseline at 10 Gbps, while the uncompressed decentralized configuration converges substantially slower.

D.2 THROUGHPUT GAIN AS A FUNCTION OF BANDWIDTH

To characterize the regime in which activation compression provides the greatest benefit, we measure the throughput gain of ResBM relative to the uncompressed baseline across a range of inter-node bandwidth settings spanning consumer-grade to datacenter-grade links. Figure D.2 plots this gain factor for both the AdamW and Muon ResBM variants at the four bandwidths we measured: 80 Mbps, 800 Mbps, 1 Gbps and 10 Gbps.

At the three higher bandwidths (800 Mbps–10 Gbps), both ResBM variants achieve throughput close to that of the uncompressed AdamW reference, while the large throughput gains emerge at 80 Mbps. The gain remains flat across this regime because GPU compute dominates pipeline latency and the communication overhead of even the uncompressed activations is negligible. At 80 Mbps, however, the picture is very different: AdamW+ResBM achieves a $\approx$12.8× throughput gain and Muon+ResBM achieves $\approx$11.2×, both relative to the uncompressed AdamW baseline at the same bandwidth. Since we did not measure any bandwidth between 80 and 800 Mbps, our data only place the transition from the compute-bound to the communication-bound regime somewhere in that interval; its exact location is not resolved by these measurements. The slight gap between the two variants reflects Muon's additional per-step compute cost, which partially offsets the communication savings at extreme bandwidth constraints. Together, these results show that ResBM's benefit is strongly regime-dependent: it is most effective precisely in the low-bandwidth settings that define decentralized training, with negligible overhead when deployed over high-speed interconnects.

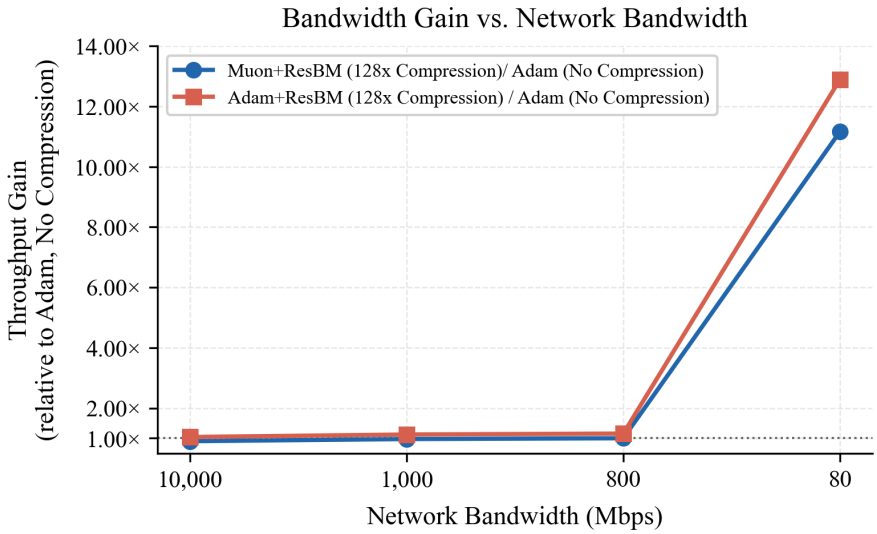


Figure D.2: Throughput gain of ResBM (128× compression) relative to the uncompressed AdamW baseline as a function of inter-node bandwidth. Gains are negligible at 800 Mbps and above, where compute dominates, and reach ≈12.8× (AdamW) and ≈11.2× (Muon) at 80 Mbps, where communication becomes the bottleneck; the transition lies somewhere between 80 and 800 Mbps.

D.3 Memory and communication overhead details

The bottleneck architecture introduces one encoder–decoder pair per pipeline boundary relative to the baseline. With the shapes of Equation 6 (encoder $H \to m \to h$, decoder $h \to m \to H$, $m = H/4$, no biases), a boundary adds $2\,(Hm + mh) = H^2/2 + Hh/2$ parameters. For the 2B model ($H = 4096$, $h = 32$) spread across 8 pipeline stages with one transformer block per stage, each of the 7 boundaries adds 8,454,144 parameters, increasing the total parameter count from 1,896,751,104 to 1,955,930,112, an overhead of approximately 59.2M parameters (3.1%).

The primary benefit lies in inter-stage communication. Without bottleneck layers, each pipeline boundary transfers, for every sequence, an $L \times H = 1024 \times 4096$ activation tensor (8 MiB in bf16) per direction. At 128× compression, this reduces to $L \times h = 1024 \times 32$ (64 KiB in bf16) — a 128× reduction in per-boundary transfer volume for both forward activations and backward gradients. Across the 7 stage boundaries in an 8-stage pipeline, and counting both directions, this translates from 112 MiB to 896 KiB per sequence; an optimizer step moves a further factor of the batch size on top of these figures. Either way the ratio is unchanged, and it is what makes the architecture practical for low-bandwidth interconnects.