COVERPIPE: A NEW COMMUNICATION PROTOCOL FOR LOW-BANDWIDTH PIPELINE PARALLELISM

Anonymous authors
Paper under double-blind review

ABSTRACT

Pipeline parallelism partitions large models across multiple accelerators to overcome single-device memory limits, but inter-stage communication poses a severe bottleneck under bandwidth-constrained networks. While recent extreme compression techniques show promise in reducing communication volume, we uncover a fundamental structural limitation: they project representations into a static, data-oblivious subspace, which severely restricts representational capacity and starves orthogonal latent dimensions of gradient updates. We formalize this failure mode under a unified analytical framework, demonstrating both theoretically and empirically that fixed projections impair model trainability. To address this issue, we propose **C**over**P**ipe, a new pipeline-parallel training scheme based on input-adaptive sparsification. **C**over**P**ipe dynamically preserves salient features per sample on the critical path, while asynchronously transmitting residual components during idle communication windows to restore full representational capacity and gradient flow. Comprehensive evaluations on standard and MoE Transformer workloads across 60–500 Mbps networks demonstrate that **C**over**P**ipe outperforms both uncompressed and compressed baselines in wall-clock convergence while maintaining high throughput. Under Chinchilla-optimal training budgets, **C**over**P**ipe consistently outperforms existing compression methods across compression ratios from $8\times$ to $128\times$, demonstrating robust long-horizon optimization. Experiments on models up to 1B parameters further confirm the robustness of **C**over**P**ipe, establishing it as an effective and practical solution for pipeline parallelism over commodity networks.

1 INTRODUCTION

The memory demands of training increasingly large models often exceed the capacity of a single accelerator. Pipeline parallelism addresses this limitation by partitioning a model into consecutive stages placed across multiple accelerators (Huang et al., 2019; Narayanan et al., 2021). Each stage executes computation sequentially, transferring intermediate activations forward and propagating the corresponding gradients backward. Because stages strictly depend on these boundary tensors before resuming computation, pipeline throughput hinges on network bandwidth. Outside specialized high-bandwidth interconnects such as commodity cross-node networks where bandwidth drops by orders of magnitude, this communication overhead heavily stalls computations and becomes a prohibitive performance bottleneck.

To alleviate this communication overhead and reduce reliance on high-bandwidth fabrics, recent studies have explored lossy compression for the intermediate tensors transferred between pipeline stages. Remarkably, these approaches demonstrate that training can remain stable even at extreme compression ratios of $100\times$ (Ramasinghe et al., 2025b; Aboudib et al., 2026). Such empirical success highlights the strong potential of boundary tensor compression to democratize pipeline-parallel training over bandwidth-constrained networks.

Our analysis, however, reveals a fundamental structural limitation shared by these methods: they project the representations of all inputs into the same static, data-oblivious subspace. This rigid projection not only constrains the representational capacity across stage boundaries, but also severely impairs the trainability of latent dimensions lying outside the preserved subspace. We formalize this phenomenon through a unified analytical framework that characterizes existing approaches as

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.

instances of *Bottleneck Compression*, and show that the backward pass fails to propagate sufficient gradients to the orthogonal subspace. Our empirical analysis further confirms this theoretical insight, demonstrating that dimensions orthogonal to the preserved subspace receive substantially weaker gradient signals.

To overcome this structural limitation, we introduce **C**over**P**ipe, a novel communication protocol for communication-efficient pipeline-parallel training. Instead of projecting representations into a rigid, static subspace, **C**over**P**ipe performs input-adaptive sparsification, dynamically preserving the most salient features for each input sample. Crucially, the residual information from the unselected dimensions is transmitted asynchronously during idle communication windows and merged with the primary activations at subsequent stages. By decoupling critical-path transfers from opportunistic residual transmission, **C**over**P**ipe preserves representational capacity and restores vital gradient flow across all latent dimensions, enabling stable and efficient training under aggressive compression.

Extensive empirical evaluations validate the efficiency, robustness, and scalability of **C**over**P**ipe across diverse Transformer workloads. Across standard and MoE Transformers, **C**over**P**ipe achieves better model quality within the same wall-clock budget under constrained network bandwidths (60–500 Mbps) than both uncompressed training and existing compression methods. This advantage is preserved under Chinchilla-optimal training budgets (Hoffmann et al., 2022) across compression ratios from $8\times$ to $128\times$, demonstrating the robustness of **C**over**P**ipe over long training horizons. Finally, evaluations on models up to 1B parameters demonstrate that **C**over**P**ipe is an effective, practical solution for pipeline parallelism over low-bandwidth networks.

2 Problem Diagnosis

In this section, we analyze the structural limitations of recent communication-efficient pipeline parallelism methods from a unified viewpoint that contrasts uncompressed and compressed training dynamics. For simplicity, we assume each pipeline stage contains m sequential residual blocks.

Notation. We use $I_d \in \mathbb{R}^{d\times d}$ to denote the identity matrix and $I_{d_1\times d_2} \in \mathbb{R}^{d_1\times d_2}$ to denote the rectangular matrix whose (i,j)-th entry is 1 if $i=j$ and 0 otherwise. We write $\{a,\dots,b\}$ to denote $\{j\in\mathbb{Z} : a\le j\le b\}$. For an ordered index set $S=\{j_1<\cdots<j_q\}$, we define $\prod_{j\in S} M_j := M_{j_q}\cdots M_{j_1}$. Finally, the Jacobian of function $f(\mathbf{x})$ at $\mathbf{x}$ is $J_f(\mathbf{x}) := \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$.

2.1 Standard Pipeline Parallelism

We begin with standard pipeline parallelism without communication compression. Let $\mathbf{x}^{\ell-1}\in\mathbb{R}^H$ denote an output of stage $\ell-1$, where H is the hidden dimension. Without compression, the input of the first block of the next stage $\mathbf{x}_0^\ell$ is equal to $\mathbf{x}^{\ell-1}$. Let $\mathbf{x}_i^\ell$ denote the output of the i-th residual block within the stage. Then, the i-th residual block is written as

$$\mathbf{x}_i^\ell = \mathbf{x}_{i-1}^\ell + f_i^\ell\big(\mathbf{x}_{i-1}^\ell\big), \qquad i\in\{1,\dots,m\}, \tag{1}$$

where f_i^ℓ denotes the residual branch with learnable parameters. Consequently, the representations at the same block position i in two adjacent stages satisfy

$$\mathbf{x}_i^{\ell+1} = \mathbf{x}_i^\ell + \sum_{j=i+1}^{m} f_j^\ell\big(\mathbf{x}_{j-1}^\ell\big) + \sum_{j=1}^{i} f_j^{\ell+1}\big(\mathbf{x}_{j-1}^{\ell+1}\big). \tag{2}$$

The first term in Equation (2) forms an identity path that directly carries $\mathbf{x}_i^\ell$ to $\mathbf{x}_i^{\ell+1}$ without passing through any residual branch. This identity path is known to play a key role in training the model by providing direct gradient propagation (He et al., 2016).

To see this, applying the chain rule to Equation (2) gives

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_i^\ell} = I_H + A_{1,i}^\ell + A_{2,i}^{\ell+1} + A_{2,i}^{\ell+1}A_{1,i}^\ell, \tag{3}$$

where $A_{1,i}^\ell$ and $A_{2,i}^{\ell+1}$ are the sums of residual-branch Jacobian products taken in chain-rule order over blocks $\{i+1,\dots,m\}$ in stage ℓ and blocks $\{1,\dots,i\}$ in stage $\ell+1$, respectively. Specifically, $A_{1,i}^\ell = \sum_{\emptyset\neq S\subseteq\{i+1,\dots,m\}} \prod_{j\in S} J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell)$ and $A_{2,i}^{\ell+1} = \sum_{\emptyset\neq S\subseteq\{1,\dots,i\}} \prod_{j\in S} J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1})$.

The repeated multiplication of Jacobian terms in $A^{\ell}_{1,i}$ and $A^{\ell+1}_{2,i}$ causes the gradient magnitude to be susceptible to decay, leading to the vanishing gradient problem. However, the identity term I_H provides direct gradient propagation across all H hidden dimensions. Consequently, the addition of I_H mitigates this signal decay, allowing residual networks to remain trainable at large depths.

2.2 Bottleneck Compression

We next show that recent communication-efficient pipeline parallelism methods, despite their architectural differences, share a common bottleneck structure. Let $\mathbf{b}^{\ell-1} \in \mathbb{R}^h$ denote the compressed $\mathbf{x}^{\ell-1}$ transmitted to the next stage.

Subspace Networks (SSN) (Ramasinghe et al., 2025b) use an orthonormal basis $U \in \mathbb{R}^{H\times h}$ and reconstruct the hidden representation as

$$\mathbf{x}^{\ell}_0 = U\mathbf{b}^{\ell-1} + \mathbf{s}, \tag{4}$$
$$\mathbf{x}^{\ell}_i = \mathbf{x}^{\ell}_{i-1} + f^{\ell}_i\big(\mathbf{x}^{\ell}_{i-1}\big), \qquad i \in \{1,\dots,m\}, \tag{5}$$
$$\mathbf{b}^{\ell} = U^{\top}\big(\mathbf{x}^{\ell}_m - \mathbf{s}\big), \tag{6}$$

where $U^{\top}U = I_h$ and $\mathbf{s} \in \mathbb{R}^H$ denotes the locally reconstructible fixed high-rank component.

Residual Bottleneck Models (ResBM) (Aboudib et al., 2026), on the other hand, use learned nonlinear encoder and decoder mappings. Let $D^{\ell} : \mathbb{R}^h \to \mathbb{R}^H, C^{\ell} : \mathbb{R}^H \to \mathbb{R}^h$ denote the decoder and encoder, respectively. For ResBM, these mappings are implemented using learnable projection matrices and nonlinear activations. Its layer update can be written as

$$\mathbf{x}^{\ell}_0 = D^{\ell}(\mathbf{b}^{\ell-1}), \tag{7}$$
$$\mathbf{x}^{\ell}_1 = I_{H\times h}\mathbf{b}^{\ell-1} + f^{\ell}_1\big(\mathbf{x}^{\ell}_0\big), \tag{8}$$
$$\mathbf{x}^{\ell}_i = \mathbf{x}^{\ell}_{i-1} + f^{\ell}_i\big(\mathbf{x}^{\ell}_{i-1}\big), \qquad i \in \{2,\dots,m-1\}, \tag{9}$$
$$\mathbf{b}^{\ell} = I_{h\times H}\mathbf{x}^{\ell}_{m-1} + C^{\ell}\big(f^{\ell}_m(\mathbf{x}^{\ell}_{m-1})\big). \tag{10}$$

Although SSN and ResBM implement compression differently, both can be expressed in the common form

$$\mathbf{x}^{\ell}_1 = P\mathbf{b}^{\ell-1} + \mathbf{s} + f^{\ell}_1\big(D^{\ell}(\mathbf{b}^{\ell-1})\big), \tag{11}$$
$$\mathbf{x}^{\ell}_i = \mathbf{x}^{\ell}_{i-1} + f^{\ell}_i\big(\mathbf{x}^{\ell}_{i-1}\big), \qquad i \in \{2,\dots,m-1\}, \tag{12}$$
$$\mathbf{b}^{\ell} = P^{\top}\big(\mathbf{x}^{\ell}_{m-1} - \mathbf{s}\big) + C^{\ell}\big(f^{\ell}_m(\mathbf{x}^{\ell}_{m-1})\big), \tag{13}$$

where $P \in \mathbb{R}^{H\times h}$ has orthonormal columns, $D^{\ell} : \mathbb{R}^h \to \mathbb{R}^H$, and $C^{\ell} : \mathbb{R}^H \to \mathbb{R}^h$.

We refer to architectures of this form as *bottleneck compression*.

Remark. Assigning one Transformer layer per stage gives $m = 2$, where f^{ℓ}_1 and f^{ℓ}_2 correspond to the self-attention and feed-forward residual branches, respectively. The unified formulation recovers SSN by choosing $P = U$, $D^{\ell}(\mathbf{b}) = U\mathbf{b} + \mathbf{s}$, and $C^{\ell}(\mathbf{x}) = U^{\top}\mathbf{x}$. ResBM is obtained by setting $P = I_{H\times h}$ and $\mathbf{s} = 0$, with D^{ℓ} and C^{ℓ} given by its decoder and encoder.

2.3 Structural Limitations of Bottleneck Compression

We now examine how bottleneck compression alters the direct gradient propagation identified in Section 2.1. For each $i \in \{1,\dots,m-1\}$, applying the chain rule as in Equation (3) gives

$$\frac{\partial \mathbf{x}^{\ell+1}_i}{\partial \mathbf{x}^{\ell}_i} = PP^{\top} + A'^{\ell+1}_{2,i}P^{\top} + PA'^{\ell}_{1,i} + A'^{\ell+1}_{2,i}A'^{\ell}_{1,i}. \tag{14}$$

where $A'^{\ell}_{1,i}$ and $A'^{\ell+1}_{2,i}$ collect products of the Jacobians in chain-rule order for each stage. Compared with the full-dimensional identity I_H in Equation (3), the direct gradient propagation is governed by $\Pi := PP^{\top}$. Since Π is a projection matrix by definition, $\nabla_{\mathbf{x}^{\ell}_i}\mathcal{L} = \left(\frac{\partial \mathbf{x}^{\ell+1}_i}{\partial \mathbf{x}^{\ell}_i}\right)^{\top} \nabla_{\mathbf{x}^{\ell+1}_i}\mathcal{L}$ is decomposed by the component lies on the column space of Π and the component lies on the orthogonal

space. Specifically,

$$\mathrm{Col}(\Pi): \quad \Pi\nabla_{\mathbf{x}_i^\ell}\mathcal{L} = \Pi\nabla_{\mathbf{x}_i^{\ell+1}}\mathcal{L} + \Pi\left(A_{2,i}'^{\ell+1}P^\top + PA_{1,i}'^{\ell} + A_{2,i}'^{\ell+1}A_{1,i}'^{\ell}\right)^\top \nabla_{\mathbf{x}_i^{\ell+1}}\mathcal{L}, \tag{15}$$

$$\mathrm{Null}(\Pi): \quad (I_H-\Pi)\nabla_{\mathbf{x}_i^\ell}\mathcal{L} = (I_H-\Pi)\left(A_{2,i}'^{\ell+1}P^\top + PA_{1,i}'^{\ell} + A_{2,i}'^{\ell+1}A_{1,i}'^{\ell}\right)^\top \nabla_{\mathbf{x}_i^{\ell+1}}\mathcal{L}. \tag{16}$$

Equation (16) shows that the gradient component in $\mathrm{Null}(\Pi)$ receives no direct contribution from the identity path. Instead, it depends entirely on terms involving Jacobians, making it susceptible to gradient decay. This restriction holds at every residual-block position $i \in \{1, \ldots, m-1\}$ within each stage. Notably, because Π is fixed regardless of input, the same hidden directions repeatedly lack a direct gradient contribution. Thus, the effect of bottleneck compression extends beyond limiting representational capacity. It can also impair the trainability of hidden directions outside the preserved subspace within each stage. Detailed derivations for Equations (3) and (14) are in Appendix B.

3 New Communication Protocol

Motivated by our analysis of bottleneck compression, we propose **C**over**P**ipe, a new communication protocol that combines input-adaptive sparsification with asynchronous residual transmission. Sparse **Co**re Communication dynamically preserves salient features for each input across adjacent stages (Section 3.1), while Residual **Over**lap transmits residual information during idle communication intervals and merges it with the primary activations at a subsequent stage (Section 3.2). We present a conceptual comparison of communication scheme in Section 3.3.

3.1 Sparse **Co**re Communication

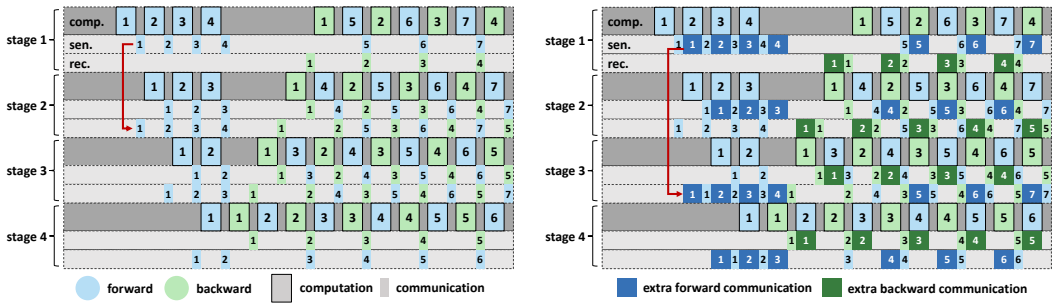
Let $\mathbf{X}^\ell = [\mathbf{x}_{(1)}^\ell, \ldots, \mathbf{x}_{(N)}^\ell] \in \mathbb{R}^{H\times N}$ denote the N sample representations produced by stage ℓ. For each sample (n), our protocol selects a core index set $\mathcal{I}_{(n)}^\ell \subseteq [H]$ important to $\mathbf{x}_{(n)}^\ell$. This dynamic selection inherently makes direct paths available to any hidden coordinate, allowing meaningful gradients to reach all hidden dimensions.

Forward. The sender constructs $\mathcal{I}_{(n)}^\ell := \mathcal{G}^\ell \cup \mathcal{S}_{(n)}^\ell$ from the shared index set $\mathcal{G}^\ell$ and the sample-specific index set $\mathcal{S}_{(n)}^\ell$. The shared set $\mathcal{G}^\ell$ captures commonly important coordinates across samples by selecting the r largest $\|\mathbf{X}_{[j,:]}^\ell\|_1$ indices. Its indices are communicated only once, incurring negligible communication overhead. To retain sample adaptivity, $\mathcal{S}_{(n)}^\ell$ contains the k indices with the largest $|x_{(n),j}^\ell|$ among $j \notin \mathcal{G}^\ell$. The selected values are rescaled to preserve the norm of sample representation: $\alpha_{(n)}^\ell := \frac{\|\mathbf{x}_{(n)}^\ell\|_2}{\left(\sum_{j\in\mathcal{I}_{(n)}^\ell}|x_{(n),j}^\ell|^2\right)^{1/2}}$.

Layer $\ell+1$ receives the information of $\{\alpha_{(n)}^\ell x_{(n),j}^\ell\}_{j\in\mathcal{I}_{(n)}^\ell}$ and $\mathcal{I}_{(n)}^\ell$. Then, it reconstructs the sparse representation by scattering the received values into the selected coordinates and fills the remaining coordinates with zero.

Backward. The coordinate set $\mathcal{I}_{(n)}^\ell$ selected during the forward pass is cached and reused during the backward pass. This enables exact backpropagation with respect to the scaled sparse representation. Specifically, the sparsification operation is locally constant except at boundaries where the ordering of the coordinate magnitudes changes. Its derivative is therefore zero almost everywhere, leading to

$$\frac{\partial\mathcal{L}}{\partial\left(\alpha_{(n)}^\ell x_{(n),j}^\ell\right)} = \frac{\partial\mathcal{L}}{\partial x_{(n),j}^{\ell+1,0}}\left(\mathbb{1}_{\{j\in\mathcal{I}_{(n)}^\ell\}} + \alpha_{(n)}^\ell x_{(n),j}^\ell \underbrace{\frac{\partial\mathbb{1}_{\{j\in\mathcal{I}_{(n)}^\ell\}}}{\partial\left(\alpha_{(n)}^\ell x_{(n),j}^\ell\right)}}_{=0 \text{ a.e.}}\right) = \frac{\partial\mathcal{L}}{\partial x_{(n),j}^{\ell+1,0}}\mathbb{1}_{\{j\in\mathcal{I}_{(n)}^\ell\}}. \tag{17}$$



(a) 1F1B scheduling

(b) 1F1B with residual overlap scheduling

Figure 1: **1F1B pipeline schedules with and without residual overlap.** Time proceeds from left to right, and numbers denote microbatch IDs. Each pipeline stage is represented by computation, send, and receive streams; blue and green denote forward and backward operations, respectively. Red arrows indicate the communication flow of the first microbatch from stage 1. (a) In standard 1F1B schedule, each stage computes sequentially, transferring activations forward and gradients backward. During this computation, communication links remain idle. (b) Extra forward (dark blue) and backward (dark green) transfers send residual information during the communication-idle intervals. Bypassing one intermediate stage allows these extra transfers to overlap with the computation of that stage. When the extra transfers fit within these link-idle intervals, they introduce no additional computation stall from communication.

Communication budget. We choose r and k to match the communication budget of h-dimension bottleneck compression. Specifically, for n_b bits values, we set k to the largest feasible value:

$$k_{\max} := \max\left\{k : kn_b + L_{\text{idx}}(k) \le (h-r)n_b\right\}, \tag{18}$$

where $L_{\text{idx}}(k) = \left\lceil \log_2 \binom{H-r}{k} \right\rceil$ is the minimum number of bits needed to encode k positions among the $H-r$ remaining coordinates. We attain this bound using enumerative coding (Cover, 1973). The exact encoding and decoding procedures and cost analysis are given in Appendix A.

3.2 Residual Overlap

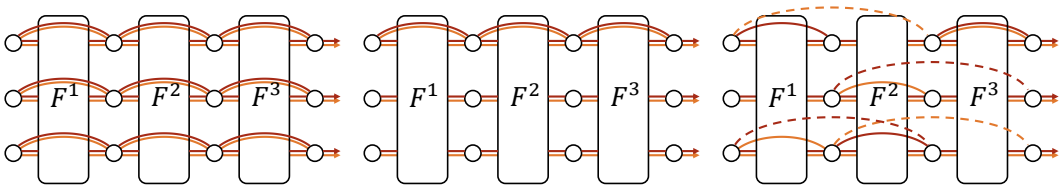
In standard pipeline parallelism, each stage must wait to receive this information before resuming the corresponding computation. This dependency limits the amount of information each stage can receive within a fixed time under low-bandwidth conditions, constraining the number of hidden directions with identity paths for each input. Residual overlap increases this number by transmitting an additional set of coordinates across one intervening stage during the communication-idle periods. For clarity, we omit the sample indices below.

Let $\mathcal{I}^\ell$ and $\mathcal{E}^\ell$ denote the primary and extra coordinate sets selected from $\mathbf{x}^\ell$, the output of stage ℓ. The primary indices $\mathcal{I}^\ell$ and their scaled values are transmitted to stage $\ell+1$ and reconstructed as $\alpha^\ell \widetilde{\mathbf{x}}^\ell$. While stage $\ell+1$ performs its forward computation, the extra indices $\mathcal{E}^\ell$ and their values are transmitted to stage $\ell+2$ and reconstructed as $\widehat{\mathbf{x}}^\ell$. Consequently, stage $\ell+1$ receives $\alpha^\ell \widetilde{\mathbf{x}}^\ell$ from stage ℓ and $\widehat{\mathbf{x}}^{\ell-1}$ from stage $\ell-1$, forming

$$\mathbf{x}_0^{\ell+1} := \alpha^\ell \widetilde{\mathbf{x}}^\ell + \widehat{\mathbf{x}}^{\ell-1}. \tag{19}$$

Backpropagation provides the gradients with respect to both $\alpha^\ell \widetilde{\mathbf{x}}^\ell$ and $\widehat{\mathbf{x}}^{\ell-1}$. Since $\widehat{\mathbf{x}}^{\ell-1}$ bypasses stage ℓ in the forward pass, its gradient does not participate in the backward computation of that stage. It can therefore be transmitted directly from stage $\ell+1$ to stage $\ell-1$ after the transmission of the primary information gradient, avoiding communication contention between gradient transfers.

Figure 1 illustrates how residual overlap exploits the communication-idle periods in standard pipeline parallelism schedules. In a standard schedule, communication links remain idle while a stage processes its forward or backward computation. Our method schedules the transmission of the extra information to overlap with the computation of the bypassed stage. Because this transmission overlaps with computation, the amount of information η transferred through residual overlap can be



(a) No Compression (b) Bottleneck Compression (c) **C**over**P**ipe

Figure 2: **Conceptual comparison of cross-stage identity paths.** Rows denote hidden directions, colors denote inputs, and curves indicate identity paths across computations F^ℓ. (a) No compression preserves all directions; (b) bottleneck compression preserves one fixed subspace; (c) **C**over**P**ipe uses input-adaptive critical paths (solid) and bypass paths overlapped with intermediate-stage computation (dashed).

chosen independently from the critical path's compression ratio. When transmission fits within the available computation window, it introduces no additional communication-induced stall.

To construct the extra set $\mathcal{E}^\ell$, we first mask the activations by setting the values at $\mathcal{I}^\ell$ to zero. We then apply the selection scheme from Section 3.1 to these masked activations and transmit the chosen values without scaling.

3.3 Comparison of Communication Schemes

To clarify the structural differences among these communication schemes, Figure 2 illustrates how each scheme propagates information and gradients across pipeline stages. Standard pipeline parallelism preserves a direct path along every hidden direction, so each coordinate receives a gradient component that does not pass through the residual branches (Figure 2a). Bottleneck compression instead restricts this path to a fixed low-dimensional projection shared across inputs and boundaries; the complementary directions must propagate through the learned transformations (Figure 2b).

Cover**P**ipe avoids repeatedly protecting the same subspace through two complementary mechanisms (Figure 2c). Sparse core communication varies the primary coordinates across inputs and boundaries, while residual overlap transmits unselected coordinates across an intervening stage during its computation. Together, the input-adaptive primary paths and longer-range bypass paths expand direct gradient coverage without adding communication stall when the extra transfer fits within the available idle interval.

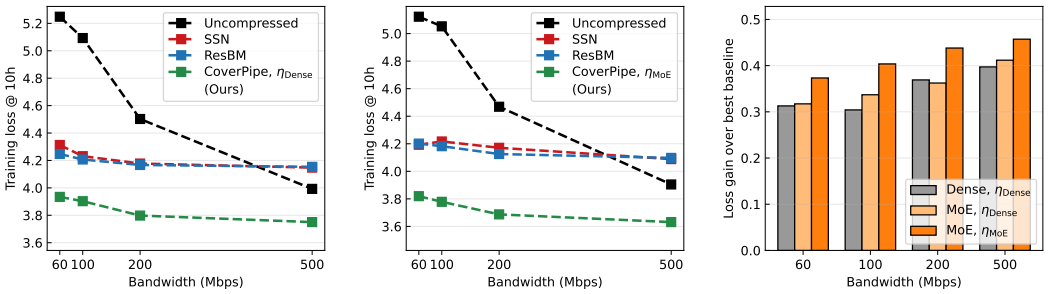
4 Experiments

Practical training under the fixed time budget requires both high throughput and strong per-step convergence. Under bandwidth-constrained networks, communication reduces throughput, motivating compression to improve wall-clock convergence. To validate the effectiveness of **C**over**P**ipe under this constraint, we first compare its wall-clock convergence against uncompressed training and communication-efficient pipeline parallel methods across diverse network bandwidths and Transformer architectures (Section 4.2). We then evaluate all methods under a fixed token budget to isolate optimization performance from the effect of throughput (Section 4.3). Finally, we provide empirical support for our motivation and validate the effectiveness of our sparsification design (Section 4.4).

Full results of each experiment and details are provided in Appendix D.1.

4.1 Experimental Setup

Model and data. We train Llama-style Transformers (Touvron et al., 2023a) on OpenWebText (Gokaslan et al., 2019). Most experiments use an 8-layer dense model with hidden dimension $H = 1024$ (200M parameters), following the common width-to-depth ratio of Transformers (Touvron et al., 2023b; Brown et al., 2020). Each pipeline stage contains one Transformer layer, with compression applied at every stage boundary.



(a) Standard Transformer (b) MoE Transformer (c) Gain over the best baseline

Figure 3: **Training under low-bandwidth conditions at $\rho = 64$.** (a) **C**over**P**ipe outperforms all baselines across every bandwidth on the dense model. (b) The advantage of **C**over**P**ipe extends to the MoE model. (c) Additional per-stage computation in the MoE setting enables $\eta_{\text{Dense}} \leq \eta_{\text{MoE}}$, resulting in a greater loss gain over the best baseline.

Communication budget and baselines. Bottleneck compressions transmit an h-dimensional BF16 representation, whereas **C**over**P**ipe allocates the same budget between values and indices on the critical path. We denote $\rho = H/h$ for this critical path compression ratio. For each bandwidth in the low-bandwidth experiments, we choose the amount of information η transferred through residual overlap so that its transmission fits within the computation window. Additional hyperparameters, exact communication budgets, and baseline implementation details are provided in Appendix C.

4.2 Results under Low-Bandwidth Constraints

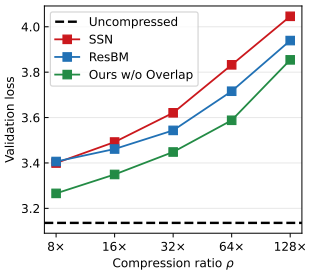
Under bandwidth-constrained networks, standard pipeline parallelism becomes increasingly inefficient as inter-stage communication prolongs each training step. Communication compression can recover throughput, but often at the cost of optimization degradation. Therefore, practical performance depends jointly on training throughput and per-step convergence to achieve optimal performance within a fixed time budget. To validate the efficiency of **C**over**P**ipe under this constraint, we first compare wall-clock convergence across diverse bandwidths.

Figure 3 presents the training loss achieved by each method within a fixed 10-hour budget. For each bandwidth, η_{Dense} and η_{MoE} denote the amounts of information carried by residual overlap per token in the standard dense and MoE models, respectively. As shown in Figure 3a, uncompressed training degrades sharply as bandwidth decreases because communication reduces the number of optimization steps completed within the budget. The compression baselines largely mitigate this throughput loss by reducing inter-stage communication, resulting in substantially lower training loss at low bandwidths (60–200 Mbps). However, the degradation in per-step convergence causes them to be overtaken by uncompressed training at 500 Mbps. In contrast, **C**over**P**ipe achieves the lowest training loss across all bandwidths, demonstrating that it effectively reduce the optimization degradation caused by communication compression.

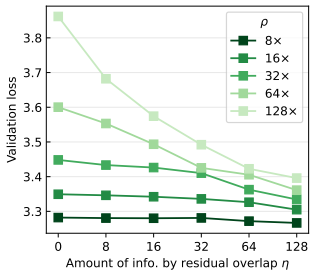
We next evaluate whether the effectiveness of **C**over**P**ipe extends to MoE Transformers. As shown in Figure 3b, the MoE model exhibits the same trend as the dense model, with **C**over**P**ipe achieving the lowest training loss across all bandwidths. Notably, processing two selected experts per layer creates a longer computation window for residual overlap, allowing the MoE model to use larger η_{MoE} than η_{Dense} by overlapping. Figure 3c shows this benefit from larger computation window. The loss gain of **C**over**P**ipe with ρ_{Dense} over best baseline on MoE training is comparable to that of dense training. Choosing η_{MoE} for MoE training yields a greater loss reduction, widening the advantage of **C**over**P**ipe in models with greater computational workloads.

4.3 Results at a Fixed Token Budget

The wall-clock results in Section 4.2 reflect the combined effects of throughput and per-step convergence. To isolate the effect of compression on per-step convergence and assess the separate contributions of sparsification and overlap, we next compare methods under a fixed token budget.



(a) Final loss across ρ ($\eta = 0$)



(b) Residual overlap

Figure 4: **Performance across compression ratios.** Loss after Chinchilla-optimal token budget. (a) **C**over**P**ipe without residual overlap alone outperforms all baselines across every ρ. (b) The benefit of residual overlap grows as η increases and becomes more significant as ρ increases.

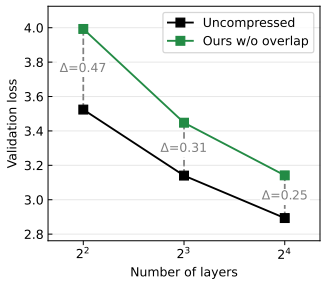


Figure 5: **Effect of model scale.** Loss vs. model depth for uncompressed and **C**over**P**ipe without residual overlap. The gap narrows as model scale increases.

Figure 4 demonstrates the effectiveness of **C**over**P**ipe under a Chinchilla-optimal token budget. As illustrated in Figure 4a, **C**over**P**ipe without residual overlap alone outperforms all compression baselines across compression ratios ρ from $8\times$ to $128\times$. However, its performance gap to uncompressed training widens as ρ increases, becoming pronounced under extreme compression. Introducing residual overlap narrows this gap, as demonstrated in Figure 4b. At each ρ, validation loss monotonically decreases as η increases. Further, the decrease is larger at higher ρ, suggesting its greater benefit under extreme compression.

We next examine how the performance gap between **C**over**P**ipe and uncompressed training changes with model scale. We double the depth from 4 to 8 to 16 layers (65M to 1B parameters), preserving the width-to-depth ratio. Figure 5 shows that **C**over**P**ipe is stable under model scaling and that the gap narrows as model size increases. Notably, this gap narrows even without residual overlap. By the square-cube law (Ryabinin et al., 2023), computation grows faster than activation communication as the model scales, allowing residual overlap to transfer more information during the computation periods. This suggests that **C**over**P**ipe could further reduce the compression penalty at larger scales, highlighting its potential as a practical solution for pipeline parallelism over commodity networks.

4.4 Ablations

Effect of sparsification design. We compare three sparsification variants: independent forward/backward index selection (Ramasinghe et al., 2025a), backward matching (Bian et al., 2024; Rudakov et al., 2023), and our sparsification design with index compression and norm-preserving rescaling. Figure 6 shows that independent index selection causes training to fail, whereas backward matching, which reuses the forward-selected indices during the backward pass, restores stable optimization. Index compression and rescaling further improve performance beyond backward matching, enabling our full design to substantially outperform both sparsification baselines.

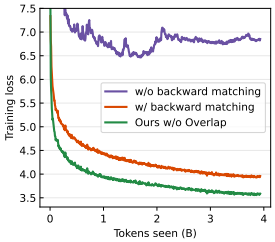


Figure 6: Training loss

Effect of CoverPipe on the gradient signal. Section 2 suggests that bottleneck compression can impair model trainability by restricting identity paths to a fixed subspace. We validate this by measuring the activation gradient magnitude across hidden directions. Figure 7 presents the gradient with respect to the output of the attention block in the fifth stage. Under uncompressed training, the gradient magnitude remains relatively uniform across the entire hidden space. In contrast, the bottleneck compressions exhibit a sharp disparity between subspaces. **C**over**P**ipe avoids this subspace-dependent suppression and maintains non-negligible gradient signals throughout the hidden space.

More experimental results. We further analyze index-encoding effect, throughput at different bandwidths, and validate **C**over**P**ipe on a wider 2B-parameter model; see Appendix D.2 for details.

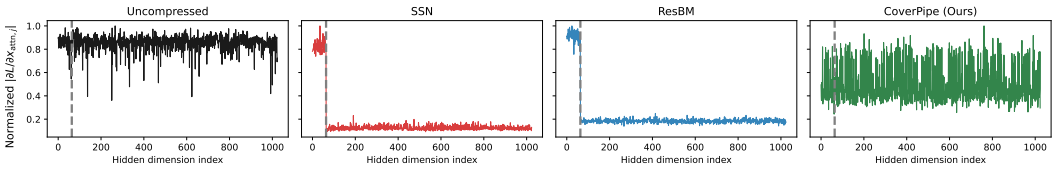


Figure 7: Magnitude of activation gradient across hidden directions

5 Other Related Works

Sparsification in pipeline parallelism Prior to SSN and ResBM, several studies explored activation compression to alleviate communication bottlenecks in pipeline parallelism (Bian et al., 2024; Rudakov et al., 2023). These studies investigated sparsification but considered relatively high-bandwidth settings and were therefore limited to moderate compression ratios. Crucially, their evaluations on language models focused on fine-tuning, where sparsification caused substantial degradation and was consequently deemed unsuitable. In contrast, we reinterpret sparsification through the lens of preserving identity paths and complement it with residual overlap. We further mitigate training instability and communication overhead via scaling and index compression. To our knowledge, this work is the first to demonstrate that sparsification can enable extreme activation compression during language-model pretraining.

Computation skipping Residual architectures are known to tolerate the omission of certain layer computations (Veit et al., 2016; Huang et al., 2016), which has been exploited for regularization and efficient inference (Fan et al., 2020). SkipPipe (Blagoev et al., 2025) extends this property to pipeline parallelism by assigning each microbatch an execution path that skips selected pipeline stages. However, because it still transmits full activations between the executed stages, accommodating lower bandwidth requires skipping more stages. In contrast, our method directly compresses activations and schedules complementary information to bypass intermediate stages during communication-idle intervals, thereby restoring the identity paths lost to compression. This design enables communication-efficient pipeline-parallel training under low-bandwidth conditions.

Approaches to communication efficiency A broad range of methods has been developed to reduce communication overhead in data-parallel training. These include compression (Wangni et al., 2018; Lin et al., 2018; Alistarh et al., 2017; Wang et al., 2018; Vogels et al., 2019; Zhao et al., 2025), communication-computation overlapping (Agarwal & Duchi, 2011; Zheng et al., 2017; Nabli et al., 2025), and local optimization methods (McMahan et al., 2017; Douillard et al., 2024). These techniques leverage the computational independence of workers. Pipeline parallelism, in contrast, imposes sequential dependencies, rendering existing overlapping and local optimization techniques difficult to apply directly. To overcome this constraint, our method combines communication compression with communication–computation overlap through a novel scheduling strategy. We thereby introduce a new pipeline-parallel training framework that achieves high communication efficiency.

6 Conclusion

In this work, we study communication-efficient pipeline-parallel training under bandwidth-constrained networks. We identify a structural limitation of existing bottleneck compression methods: their fixed, data-agnostic projections restrict identity paths to a low-dimensional subspace, weakening direct gradient propagation in the remaining hidden directions. To address this limitation, we introduce **C**over**P**ipe, which combines input-adaptive dynamic sparsification with asynchronous transmission of residual information during communication-idle periods. Our experiments demonstrate that **C**over**P**ipe achieves strong wall-clock convergence across diverse network bandwidths and model architectures, as well as state-of-arts per-step convergence across a wide range of compression ratios. These results establish **C**over**P**ipe as an effective communication scheme to pipeline-parallel training over commodity networks.

AI USE STATEMENT

In this work, we used generative AI tools for polishing writing, literature search, code implementation and extending mathematical claims. We have not used generative AI tools for drafting sections, research ideation, and generating synthetic datasets are not applicable to this work. We have reviewed all AI-assisted work. For example, AI tools were used to refine English phrasing after the text was fully drafted by the author. All suggestions to discover relevant literature was checked to avoide hallucinations. Furthermore, AI-assisted code for low-bandwidth simulations and CUDA kernels were tested and verified for correctness by the author. Lastly, AI helped generalize the mathematical proof in Appendix B by extending the authors initial derivation to a general setting, and was validated by the author. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies a communication protocol for training language models. Our experiments follow standard language model pretraining, and do not raise questions regarding the Code of Ethics.

REPRODUCIBILITY STATEMENT

Detailed procedure of our communication protocol including implementation details are described in Section 3 and Appendix A. Analytical derivations are provided in Appendix B. The experimental setup, communication budgets, and full results are reported in Appendices C and D.1. We will release the source code upon publication.

REFERENCES

Alan Aboudib, Rodrigo Lopez Portillo A., Kalei Brady, and Steffen Cruz. Resbm: Residual bottleneck models for low-bandwidth pipeline parallelism, 2026. URL https://arxiv.org/abs/2604.11947.

Alekh Agarwal and John Duchi. Distributed delayed stochastic optimization. In *NeurIPS*, 2011.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NeurIPS*, 2017.

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.

Song Bian, Dacheng Li, Hongyi Wang, Eric P. Xing, and Shivaram Venkataraman. Does compressing activations help model parallel training? In *MLSYS*, 2024.

Nikolay Blagoev, Lydia Yiyu Chen, and Oğuzhan Ersoy. Skippipe: Partial and reordered pipelining framework for training llms in heterogeneous networks, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

T. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, 1973.

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, 2024.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *ICLR*, 2020.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 630–645, 2016.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *NeurIPS*, 2022.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

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. *NeurIPS*, 2019.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

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://kellerjordan.github.io/posts/muon/`.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, 2017.

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 *NeurIPS*, 2025.

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*. Association for Computing Machinery, 2021.

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. Beyond top-k: Structured sparsification for compression in pipeline parallel. In *ICLR 2025 Workshop on Modularity for Collaborative, Decentralized, and Continual Deep Learning*, 2025a.

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 (NeurIPS)*, 2025b.

M. I. Rudakov, A. N. Beznosikov, Ya. A. Kholodov, and A. V. Gasnikov. Activations and gradients compression for model-parallel training. *Doklady Mathematics*, 108(S2):S272–S281, December 2023. ISSN 1531-8362. doi: 10.1134/s1064562423701314.

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. Swarm parallelism: Training large models can be surprisingly communication-efficient, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *NeurIPS*, 2019.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary B. Charles, Dimitris S. Papailiopoulos, and Stephen Wright. Atomo: Communication-efficient learning via atomic sparsification. In *NeurIPS*, 2018.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *NeurIPS*, 2018.

Hanzhen Zhao, Xingyu Xie, Cong Fang, and Zhouchen Lin. SEPARATE: A simple low-rank projection for gradient compression in modern large-scale model training process. In *ICLR*, 2025.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *ICML*, 2017.

CONTENTS

1 Introduction **1**

2 Problem Diagnosis **2**
2.1 Standard Pipeline Parallelism 2
2.2 Bottleneck Compression 3
2.3 Structural Limitations of Bottleneck Compression 3

3 New Communication Protocol **4**
3.1 Sparse **Co**re Communication 4
3.2 Residual **Over**lap 5
3.3 Comparison of Communication Schemes 6

4 Experiments **6**
4.1 Experimental Setup 6
4.2 Results under Low-Bandwidth Constraints 7
4.3 Results at a Fixed Token Budget 7
4.4 Ablations 8

5 Other Related Works **9**

6 Conclusion **9**

A Method Details **14**
A.1 Enumerative Coding 14
A.2 Cost Analysis 16

B Detailed Derivations for Section 2 **16**
B.1 Standard Pipeline Parallelism 17
B.2 Bottleneck Compression 17

C Experimental Details **18**
C.1 Model, Data, and Optimization 18
C.2 Communication Budgets 19
C.3 Baseline Implementation 20
C.4 Model Scale and MoE Configurations 20

D Additional Results **21**
D.1 Full Results and Details of Section 4 21
D.2 More experiment results 23

A METHOD DETAILS

In this section, we provide detailed implementation and memory/time cost analysis of our methods, omitted in Section 3.

A.1 ENUMERATIVE CODING

In this section, we describe the encoding and decoding procedures used in Section 3.1 to represent k selected token-specific indices among $H - r$ coordinates using $L_{\text{idx}}(k) = \left\lceil \log_2 \binom{H-r}{k} \right\rceil$ bits.

Encoding. Let a token representation $\mathbf{x} \in \mathbb{R}^H$ have a token-specific index set $\mathcal{S}$ selected from the complement of the shared index set $\mathcal{G}$. We set $k = k_{\max}$, where

$$k_{\max} := \max \{k : kn_b + L_{\text{idx}}(k) \leq (h - r)n_b\} . \tag{20}$$

Encoding $\mathcal{S}$ into $L_{\text{idx}}(k)$ bits therefore keeps **C**over**P**ipe within the communication budget of an h-dimensional bottleneck baseline.

Write the selected set in increasing order as

$$\mathcal{S} = \{s_1 < \cdots < s_k\}. \tag{21}$$

The encoder first constructs a reduced coordinate system by removing the r shared coordinates in $\mathcal{G}$ from the original H coordinates and relabeling the remaining $H - r$ coordinates in their original order. Under this reduced coordinate system, each s_j is mapped to

$$c_j := s_j - \sum_{g \in \mathcal{G}} \mathbb{1}\{g < s_j\}, \qquad j = 1, \ldots, k. \tag{22}$$

Thus, c_j is obtained by shifting s_j left by the number of shared coordinates preceding it. Since $\mathcal{S} \cap \mathcal{G} = \emptyset$, this mapping preserves the ordering of the selected coordinates and yields

$$0 \leq c_1 < \cdots < c_k < H - r. \tag{23}$$

The encoder assigns the reduced index set $\{c_1, \ldots, c_k\}$ its co-lexicographic rank

$$R(\mathcal{S}; \mathcal{G}) := \sum_{j=1}^{k} \binom{c_j}{j}. \tag{24}$$

This ranking defines a bijection between the k-subsets of the $H - r$ reduced coordinates and the integers

$$0, \ldots, \binom{H-r}{k} - 1. \tag{25}$$

Therefore, the rank uniquely represents $\mathcal{S}$ using $L_{\text{idx}}(k) = \left\lceil \log_2 \binom{H-r}{k} \right\rceil$ bits.

We reduce the computation cost for calculating the rank by defining precomputed table

$$T[a, j] := \binom{a}{j}, \qquad 0 \leq a \leq H - r, \quad 0 \leq j \leq k. \tag{26}$$

The table is constructed using Pascal's recurrence

$$T[a, 0] = 1, \qquad T[a, j] = T[a - 1, j] + T[a - 1, j - 1], \tag{27}$$

with $T[a, j] = 0$ for $j > a$. Since T depends only on $H - r$ and k, it is constructed once and reused across all tokens with the same compression configuration. The encoder can then compute the rank using table lookups:

$$R(\mathcal{S}; \mathcal{G}) = \sum_{j=1}^{k} T[c_j, j]. \tag{28}$$

Decoding. The decoder uses the same precomputed binomial table T to recover the reduced indices from the received rank. It initializes the remainder as

$$Q_k := R(\mathcal{S}; \mathcal{G}). \tag{29}$$

For $j = k, \ldots, 1$, the decoder finds

$$c_j := \max\{c \in \{j-1, \ldots, H-r-1\} : T[c, j] \le Q_j\}. \tag{30}$$

This search identifies the largest possible contribution of the form $\binom{c_j}{j}$ contained in the current remainder. The decoder then updates the remainder as

$$Q_{j-1} := Q_j - T[c_j, j]. \tag{31}$$

Repeating this procedure for all j recovers the unique ordered reduced index set

$$0 \le c_1 < \cdots < c_k < H - r, \tag{32}$$

and leaves $Q_0 = 0$.

The decoder then restores the original hidden-coordinate indices by reinserting the shared coordinates in $\mathcal{G}$. Let

$$\{q_0 < \cdots < q_{H-r-1}\} := \{0, \ldots, H-1\} \setminus \mathcal{G} \tag{33}$$

denote the ordered complement of $\mathcal{G}$. The original token-specific indices are recovered as

$$s_j := q_{c_j}, \qquad j = 1, \ldots, k. \tag{34}$$

Finally, the decoder scatters the received shared values into the coordinates in $\mathcal{G}$ and the token-specific values into the recovered coordinates in $\mathcal{S} = \{s_1, \ldots, s_k\}$, while setting all unselected coordinates to zero.

Wire-aligned packet representation. The $L_{\mathrm{idx}}(k)$-bit rank need not align with the activation datatype used for communication. Since one communication slot contains n_b bits, where $n_b = 16$ for FP16 or BF16 and $n_b = 32$ for FP32, the rank occupies

$$m(k) := \left\lceil \frac{L_{\mathrm{idx}}(k)}{n_b} \right\rceil \tag{35}$$

wire slots. The rank is represented by $m(k)$ little-endian n_b-bit words $(w_0, \ldots, w_{m(k)-1})$ such that

$$R(\mathcal{S}; \mathcal{G}) = \sum_{\ell=0}^{m(k)-1} w_\ell 2^{\ell n_b}, \qquad 0 \le w_\ell < 2^{n_b}. \tag{36}$$

Each n_b-bit rank word is stored in one communication slot without changing its bit pattern and is recovered by the inverse operation at the receiver.

For each token, the sender constructs the packet

$$\left[\underbrace{\mathbf{x}[\mathcal{G}]}_{r \text{ shared values}} \,\middle|\, \underbrace{(x_{s_1}, \ldots, x_{s_k})}_{k \text{ token-specific values}} \,\middle|\, \underbrace{(w_0, \ldots, w_{m(k)-1})}_{m(k) \text{ rank words}} \right]. \tag{37}$$

The shared index set $\mathcal{G}$ is transmitted separately and is not repeated in each token packet. The packet therefore occupies

$$r + k + m(k) \tag{38}$$

communication slots. The condition used to select k can equivalently be written as

$$k + m(k) \le h - r, \tag{39}$$

which guarantees

$$r + k + m(k) \le h. \tag{40}$$

Thus, the wire-aligned packet remains within the communication budget of an h-dimensional bottleneck baseline.

CUDA-parallel encoding and decoding. We implement the encoding and decoding procedures in custom CUDA kernels. After flattening the batch and sequence dimensions, each CUDA thread independently processes one token representation. All rank computations and table lookups for encodign/decoding kernel remain on the GPU, avoiding intermediate CPU transfers or synchronization.

A.2 Cost Analysis

Memory overhead of binomial table. A single n_b-bit word may be insufficient to store a binomial coefficient used by the encoder and decoder. We therefore represent each table entry $T[a, j]$ using $m(k)$ words of n_b bits each.

The table contains one entry for every pair $a \in \{0, \dots, H - r\}$ and $j \in \{0, \dots, k\}$, resulting in $(H - r + 1)(k + 1)$ entries. Thus, its GPU memory footprint is

$$M_{\text{table}} = \frac{n_b}{8}(H - r + 1)(k + 1)m(k) \tag{41}$$

bytes.

Table 1: Memory overhead of one binomial table under our experimental setting ($H = 1024$, BF16 communication with $n_b = 16$, and $r = \lfloor 0.6h \rfloor$).

ρ	h	r	k	$L_{\text{idx}}(k)$	$m(k)$	M_{table} (MB)
8	128	76	37	222	14	1.010
16	64	38	18	127	8	0.300
32	32	19	8	65	5	0.091
64	16	9	4	36	3	0.030
128	8	4	2	19	2	0.012

Table 1 reports the resulting memory overhead under our experimental configurations. The largest table, used at an $8\times$ compression ratio, occupies less than 1.01 MB per GPU. This corresponds to approximately 0.004% of the memory capacity of a 24-GB GPU and is independent of the batch size and sequence length. Therefore, keeping the table resident on the GPU introduces negligible memory overhead.

Computational overhead. For each token, encoding computes the rank by adding k entries from the precomputed binomial table. Since each entry consists of $m(k)$ words, the per-token encoding cost is

$$O(km(k)) \,. \tag{42}$$

Decoding recovers each of the k reduced indices by binary search over at most $H - r$ candidates. Each comparison and subtraction operates on at most $m(k)$ words, resulting in a per-token decoding cost of

$$O(km(k) \log(H - r)) \,. \tag{43}$$

For fixed ρ, n_b, and r, both k and $m(k)$ scale linearly with the hidden dimension H. Consequently, the per-token complexity of enumerative encoding is $O(H^2)$, while that of decoding is $O(H^2 \log H)$ due to the binary searches over the reduced coordinate space.

Compare to the bottleneck compression which requires $O(Hh) = O(H^2)$ operations for the projection, our method introduces no higher-order polynomial dependence on H, with only an additional logarithmic factor in decoding.

B Detailed Derivations for Section 2

In this section, we provide detailed derivations of Equations (3) and (14).

B.1 Standard Pipeline Parallelism

Since standard pipeline parallelism directly transmits $\mathbf{x}_m^\ell$ as the input $\mathbf{x}_0^{\ell+1}$ to the next stage, the cross-stage Jacobian can be decomposed as

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_i^\ell} = \frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_0^{\ell+1}} \frac{\partial \mathbf{x}_m^\ell}{\partial \mathbf{x}_i^\ell} \tag{44}$$

$$= \left(\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_{i-1}^{\ell+1}} \cdots \frac{\partial \mathbf{x}_2^{\ell+1}}{\partial \mathbf{x}_1^{\ell+1}} \frac{\partial \mathbf{x}_1^{\ell+1}}{\partial \mathbf{x}_0^{\ell+1}} \right) \left(\frac{\partial \mathbf{x}_m^\ell}{\partial \mathbf{x}_{m-1}^\ell} \cdots \frac{\partial \mathbf{x}_{i+2}^\ell}{\partial \mathbf{x}_{i+1}^\ell} \frac{\partial \mathbf{x}_{i+1}^\ell}{\partial \mathbf{x}_i^\ell} \right). \tag{45}$$

Each residual block satisfies $\frac{\partial \mathbf{x}_j^\ell}{\partial \mathbf{x}_{j-1}^\ell} = I_H + J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell)$. Therefore, for the second factor in Equation (45), the following holds:

$$\frac{\partial \mathbf{x}_m^\ell}{\partial \mathbf{x}_i^\ell} = \prod_{j \in \{i+1,\ldots,m\}} \left(I_H + J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell) \right) \tag{46}$$

$$= I_H + \sum_{\emptyset \neq S \subseteq \{i+1,\ldots,m\}} \prod_{j \in S} J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell) \tag{47}$$

$$= I_H + A_{1,i}^\ell. \tag{48}$$

Similarly, for the first factor, the following holds:

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_0^{\ell+1}} = \prod_{j \in \{1,\ldots,i\}} \left(I_H + J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1}) \right) \tag{49}$$

$$= I_H + \sum_{\emptyset \neq S \subseteq \{1,\ldots,i\}} \prod_{j \in S} J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1}) \tag{50}$$

$$= I_H + A_{2,i}^{\ell+1}. \tag{51}$$

Substituting Equations (48) and (51) into Equation (45) yields

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_i^\ell} = \left(I_H + A_{2,i}^{\ell+1} \right) \left(I_H + A_{1,i}^\ell \right) \tag{52}$$

$$= I_H + A_{1,i}^\ell + A_{2,i}^{\ell+1} + A_{2,i}^{\ell+1} A_{1,i}^\ell, \tag{53}$$

which gives Equation (3).

B.2 Bottleneck Compression

Under bottleneck compression, the representation transmitted from stage ℓ to stage $\ell+1$ is $\mathbf{b}^\ell$. Therefore, the cross-stage Jacobian can be decomposed as

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_i^\ell} = \frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{b}^\ell} \frac{\partial \mathbf{b}^\ell}{\partial \mathbf{x}_i^\ell} \tag{54}$$

$$= \left(\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_{i-1}^{\ell+1}} \cdots \frac{\partial \mathbf{x}_2^{\ell+1}}{\partial \mathbf{x}_1^{\ell+1}} \frac{\partial \mathbf{x}_1^{\ell+1}}{\partial \mathbf{b}^\ell} \right) \left(\frac{\partial \mathbf{b}^\ell}{\partial \mathbf{x}_{m-1}^\ell} \frac{\partial \mathbf{x}_{m-1}^\ell}{\partial \mathbf{x}_{m-2}^\ell} \cdots \frac{\partial \mathbf{x}_{i+1}^\ell}{\partial \mathbf{x}_i^\ell} \right). \tag{55}$$

At the compression boundary, Equation (13) gives

$$\frac{\partial \mathbf{b}^\ell}{\partial \mathbf{x}_{m-1}^\ell} = P^\top + J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell). \tag{56}$$

Here, $\circ$ is used to denote function composition. Combining this boundary Jacobian with those of the residual blocks following block i in stage ℓ gives

$$\frac{\partial \mathbf{b}^\ell}{\partial \mathbf{x}_i^\ell} = \left(P^\top + J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell)\right) \left(\prod_{j\in\{i+1,\ldots,m-1\}} \left(I_H + J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell)\right)\right) \tag{57}$$

$$= P^\top + J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell) + \sum_{\emptyset \neq S \subseteq \{i+1,\ldots,m-1\}} \left(P^\top + J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell)\right) \left(\prod_{j\in S} J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell)\right) \tag{58}$$

$$= P^\top + A'^{\ell}_{1,i}, \tag{59}$$

where

$$A'^{\ell}_{1,i} = J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell) + \sum_{\emptyset \neq S \subseteq \{i+1,\ldots,m-1\}} \left(P^\top + J_{C^\ell \circ f_m^\ell}(\mathbf{x}_{m-1}^\ell)\right) \left(\prod_{j\in S} J_{f_j^\ell}(\mathbf{x}_{j-1}^\ell)\right). \tag{60}$$

At the input boundary of stage $\ell+1$, Equation (11) gives

$$\frac{\partial \mathbf{x}_1^{\ell+1}}{\partial \mathbf{b}^\ell} = P + J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell). \tag{61}$$

Combining this boundary Jacobian with those of the residual blocks from block 2 through block i in stage $\ell+1$ gives

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{b}^\ell} = \left(\prod_{j\in\{2,\ldots,i\}} \left(I_H + J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1})\right)\right) \left(P + J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell)\right) \tag{62}$$

$$= P + J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell) + \sum_{\emptyset \neq S \subseteq \{2,\ldots,i\}} \left(\prod_{j\in S} J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1})\right) \left(P + J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell)\right) \tag{63}$$

$$= P + A'^{\ell+1}_{2,i}, \tag{64}$$

where

$$A'^{\ell+1}_{2,i} = J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell) + \sum_{\emptyset \neq S \subseteq \{2,\ldots,i\}} \left(\prod_{j\in S} J_{f_j^{\ell+1}}(\mathbf{x}_{j-1}^{\ell+1})\right) \left(P + J_{f_1^{\ell+1} \circ D^{\ell+1}}(\mathbf{b}^\ell)\right). \tag{65}$$

Substituting Equations (59) and (64) into Equation (55) yields

$$\frac{\partial \mathbf{x}_i^{\ell+1}}{\partial \mathbf{x}_i^\ell} = \left(P + A'^{\ell+1}_{2,i}\right)\left(P^\top + A'^{\ell}_{1,i}\right) \tag{66}$$

$$= PP^\top + A'^{\ell+1}_{2,i}P^\top + PA'^{\ell}_{1,i} + A'^{\ell+1}_{2,i}A'^{\ell}_{1,i}, \tag{67}$$

which gives Equation (14).

C Experimental Details

This section complements Section 4.1 with the full configuration shared by all experiments.

C.1 Model, Data, and Optimization

Table 2 lists the default configuration. The architecture follows Llama (Touvron et al., 2023a) with pre-normalization (RMSNorm), rotary position embeddings, and SwiGLU feed-forward blocks. We train for 120K optimizer steps (3.9B tokens, approximately 20 tokens per parameter, following the compute-optimal budget of Hoffmann et al. (2022)); an epoch denotes 200 steps. Validation loss is

Table 2: **Training configuration.**

	Setting	Value
Architecture	Layers / stages L	8
	Compressed boundaries	7 (forward and backward)
	Hidden dimension H	1024
	Heads (head dimension)	8 (128)
	Feed-forward	SwiGLU, hidden width 2816
	Normalization / positions	Pre-norm RMSNorm / rotary embeddings
	Parameters	205.7M
Data	Corpus	OpenWebText, 10% of documents held out
	Tokenizer	GPT-2 BPE (50,257 tokens)
	Context length	1024
	Global batch	32 (32K tokens)
Optimization	Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$
	Weight decay / grad. clipping	0.01 / 1.0
	Schedule	3-epoch linear warmup, cosine decay to 10%
	Learning rate	tuned per method and ratio (grids in text)
	Steps	120K (3.9B tokens, $\approx$ 20 tokens/parameter)
	Precision	BF16 mixed precision, BF16 inter-stage communication

computed at the end of every epoch on held-out data. The learning rate is tuned separately for each model: the uncompressed model over $\{3 \times 10^{-5}, 10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$, and every compressed method including **C**over**P**ipe without residual overlap at every compression ratio over $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}\}$. For residual overlap, we reuse the optimal learning rate of **C**over**P**ipe without residual overlap at the same compression ratio, unless otherwise stated.

Each layer is one pipeline stage, so every inter-layer transfer crosses a compressed boundary. In the wall-clock experiments of Section 4.2, each stage runs on its own GPU and compressed activations and gradients are exchanged with the 1F1B schedule using 8 microbatches of 4 sequences. In the fixed token budget experiments of Section 4.3, the same 8 logical stages are executed in a single process on one GPU, with each stage keeping its own parameters, optimizer state, and gradient clipping, and with the identical compression and decompression applied at every boundary.

C.2 Communication Budgets

Table 3: **Per-token, per-boundary communication budget at $H = 1024$.** The forward budget is Hn_b/ρ bits, with $n_b = 16$. SSN and ResBM spend all of it on an $h = H/\rho$ dimensional bottleneck. sparse core communication spends it on r shared and $k_{\max}$ token-specific coordinates, at n_b bits each, plus the enumeratively coded index of the token-specific set (Section 3.1); bits that do not fill a whole n_b-bit position are left unused. Here $\lambda = 0.6$, the fraction of the budget spent on shared coordinates ($r = \lfloor \lambda h \rfloor$).

		SSN / ResBM	sparse core communication			
ρ	budget (bits)	dimensions h	r	$k_{\max}$	coordinates $r + k_{\max}$	index (bits)
8	2048	128	76	37	113	224
16	1024	64	38	18	56	128
32	512	32	19	8	27	80
64	256	16	9	4	13	48

Table 3 gives the exact allocation at each compression ratio. Including the token-specific index encoding, the per-token forward packet of sparse core communication does not exceed the hn_b-bit budget of SSN and ResBM.

For the low-bandwidth experiments, we select η at each bandwidth to keep the TPS (token-per-second) drop relative to the compression baselines within 2%. Full results of chosen η for each experiment are given in Appendix D.1.

C.3 Baseline Implementation

All baselines are implemented in our codebase and trained on the same pipeline, data, and optimizer.

SSN. Our SSN implementation follows Ramasinghe et al. (2025b), with the publicly available implementation in the Node0 (Avraham et al., 2025) codebase serving as an additional reference.[1] At every boundary, activations are projected onto an orthonormal basis $U \in \mathbb{R}^{H\times h}$ shared across all boundaries, after subtracting a fixed high-rank component $\mathbf{s}$ implemented as a frozen, randomly initialized token-embedding table. The basis is initialized randomly and updated every 500 steps by a Grassmann-manifold gradient step (step size 0.01). We swept Grassmann-manifold step sizes of $\{0.0025, 0.005, 0.01, 0.02, 0.04\}$ and observed no appreciable differences in loss. The subspace regularizer is applied to every layer whose output still crosses a compressed boundary. Recovering $\mathbf{s}$ also requires the token identity at every stage, a further 16 bits per token that we do not count against the budget of SSN, although the analogous side information of sparse core communication—the index of the token-specific coordinates—is counted in full (Table 3). At $\rho = 64$ this would increase the communication cost of SSN by 6.3%, an effective compression ratio of 60 rather than 64.

ResBM. Following Aboudib et al. (2026), the encoder C^ℓ and decoder D^ℓ are two-layer MLPs with SiLU activations ($H \to H/4 \to h$ and $h \to H/4 \to H$, no biases), fused into the feed-forward sub-block of the last layer of the sending stage and the attention sub-block of the first layer of the receiving stage, so that the h-dimensional identity path is preserved across the boundary as in Equation (10). Encoders and decoders are trained jointly with the model using the same optimizer and learning rate.

Both baselines are implemented in our codebase; the original SSN and ResBM papers do not provide accompanying code, and Aboudib et al. (2026) report the same situation for SSN. Validation of these implementations is provided in Appendix D.

C.4 Model Scale and MoE Configurations

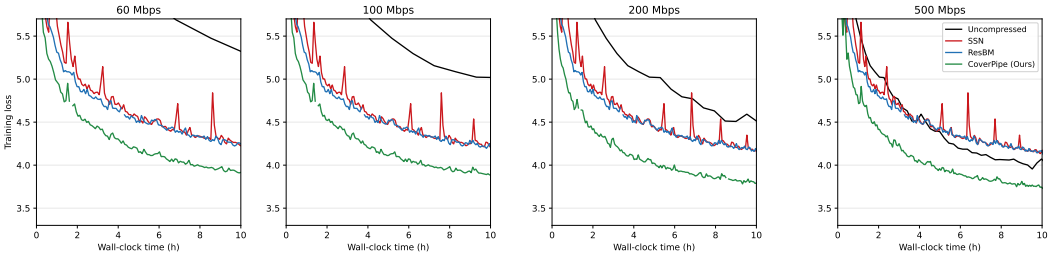
Table 4: **Model configurations used in the scaling experiments.** The head dimension (128) and the width-to-depth ratio $H/L = 128$ are held fixed; all three models are trained for the same 120K steps at $\rho = 32$.

H	L	Heads	Parameters	Boundaries
512	4	4	65.1M	3
1024	8	8	205.7M	7
2048	16	16	1028.0M	15

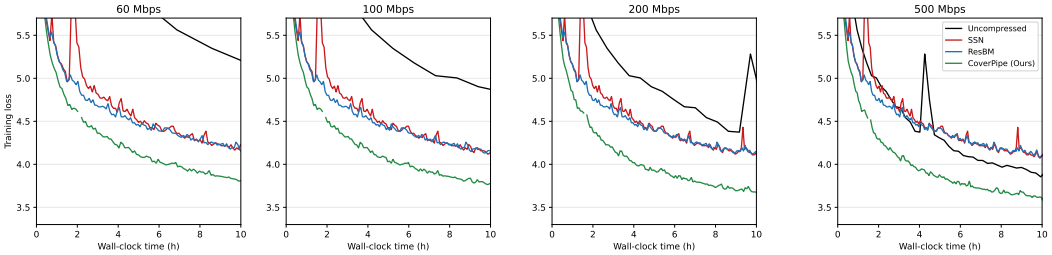
Model scale. Table 4 lists the three model sizes used in Figure 5. Every layer remains one stage, so the number of compressed boundaries grows with depth, and the fraction of each activation that is communicated between stages stays exactly $1/\rho$ at every size. Both the uncompressed and the compressed model at each size use the better of the learning rates $\{3 \times 10^{-4}, 10^{-3}\}$.

MoE. For the MoE experiments we keep the 8-layer, $H = 1024$ configuration and replace every feed-forward block with a Mixtral-style (Jiang et al., 2024) mixture of $E = 8$ experts with top-2 routing and no capacity limit (dropless). Each expert is a SwiGLU block with the same hidden width (2816) as the dense feed-forward block, so the MoE model has 690M total and 275M active parameters. Routing scores are computed in FP32, gates are the softmax over all experts renormalized over the selected two, and load balancing uses the Switch auxiliary loss with coefficient 0.02. Only the feed-forward blocks change; the residual stream that crosses pipeline boundaries, and therefore the compression, is identical to the dense model.

[1] https://github.com/PluralisResearch/node0



(a) Dense (200M).



(b) MoE (690M).

Figure 8: **Full wall-clock training curves under the 10-hour budget.** Each panel corresponds to one evaluated bandwidth (60/100/200/500 Mbps); training loss is plotted against wall-clock time to 10 hours for Uncompressed, SSN, ResBM, and **C**over**P**ipe.

D Additional Results

D.1 Full Results and Details of Section 4

Wall-clock training curves. Figure 3a and Figure 3b in Section 4.2 summarize training under a fixed 10-hour wall-clock budget. Figure 8 shows the full training curves underlying every point in those two figures, plotting training loss against wall-clock time for each of the four evaluated bandwidths (60/100/200/500 Mbps). `tokens_seen` is converted to wall-clock time via each method's measured average throughput and clipped to the 10-hour budget. Each loss-at-10h value reported in Figure 3a/Figure 3b is the mean training loss over the 20 logged epochs nearest the 10-hour mark.

Model	60 Mbps	100 Mbps	200 Mbps	500 Mbps
Dense	8	8	16	32
MoE	16	16	32	64

Table 5: Selected η (number of BF16-sized wire slots per token carried by residual overlap) for **C**over**P**ipe at each (model, bandwidth) pair.

Appendix D.1 lists the η selected among $\eta \in \{8, 16, 32, 64\}$ at each (model, bandwidth) pair using the criterion from Section 4.1: the largest η whose measured TPS stays within 2% of SSN's. MoE's selected η is always at least as large as Dense's at the same bandwidth.

Across all four bandwidths and both model sizes, **C**over**P**ipe reaches the lowest training loss within the 10-hour budget.

Fixed-token-budget training curves. Figure 4a and Figure 4b in Section 4.3 summarize training under a fixed Chinchilla-optimal token budget. Figure 9 and Figure 10 show the full training curves underlying every point in those two figures, plotting validation loss. Figure 9 has one panel per method with one curve per critical-path compression ratio $\rho \in \{8, 16, 32, 64, 128\}$. Figure 10 has one panel per ρ with one curve per swept η. Each reported loss value is the mean over the last 20 logged epochs of an EMA($\lambda = 0.9$)-smoothed curve, the same smoothing and averaging used

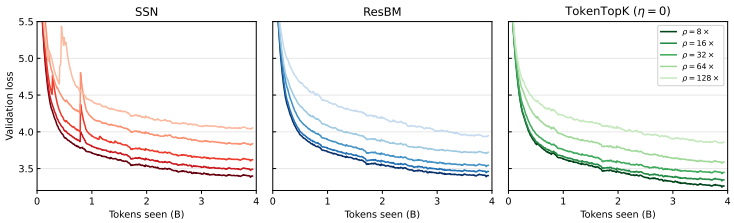


Figure 9: **Full training curves underlying Figure 4a.** Each panel corresponds to one method (SSN, ResBM, or TokenTopK without residual overlap, i.e. $\eta = 0$). We select optimal learning rate for each ρ for each method.

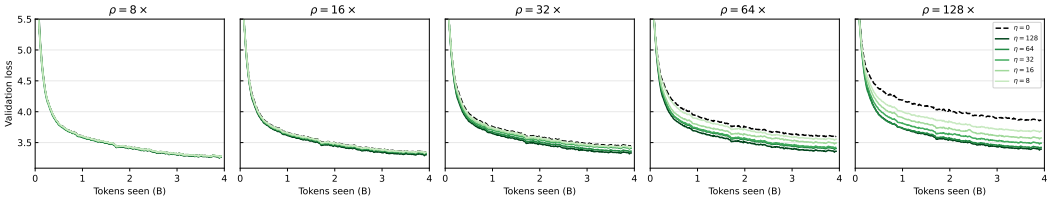
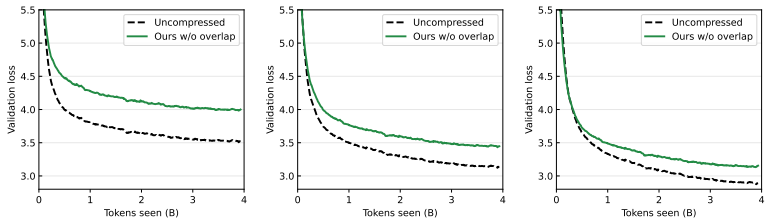


Figure 10: **Full training curves underlying Figure 4b.** Each panel corresponds to one critical-path compression ratio ρ. Learning rate is fixed to 0.001 across the sweep η, including the $\eta = 0$ baseline.

to derive every point in Section 4.3. Figure 9 uses each method's optimal learning rate at that ρ. Figure 10 fixes lr$= 0.001$ across the η sweep.

Scaling. Figure 5 in Section 4.3 summarizes the effect of model scale on communication compression. Figure 11 shows the full training curves underlying that result, plotting validation loss. Each panel is one model depth (4, 8, or 16 layers), comparing uncompressed training against our method with $\eta = 0$ at $\rho = 32$. Depth doubles from 4 to 8 to 16 layers while preserving the width-to-depth ratio. Each reported optimal-lr curve is EMA($\lambda = 0.9$)-smoothed, the same smoothing used to derive every point in Section 4.3.



(a) 4 layers. (b) 8 layers. (c) 16 layers.

Figure 11: **Full training curves underlying Figure 5.** Each panel is one model depth (4, 8, or 16 layers). The gap between compressed and uncompressed training narrows as depth increases.

Effect of sparsification design. Figure 6 in Section 4.4 compares the three sparsification variants on the 8-layer dense model at $\rho = 64$ with learning rate $= 0.003$. For backward matching (Bian et al., 2024; Rudakov et al., 2023), we store values in BF16 and indices in INT32: this baseline sparsifies at a granularity of at least sequence length $\times$ hidden dimension, so its index range exceeds what INT16 can represent.

Magnitude of activation gradient. Figure 7 in Section 4.4 shows the activation gradient magnitude at the fifth stage, using a checkpoint at epoch 100 with $\rho = 16$ and $\eta = 4$. Figure 12 shows the same normalized curves for every other stage. The gradient is normalized to peak at 1 within each (method, layer) pair. The pattern in Figure 7 holds in other layers: **C**over**P**ipe maintains non-

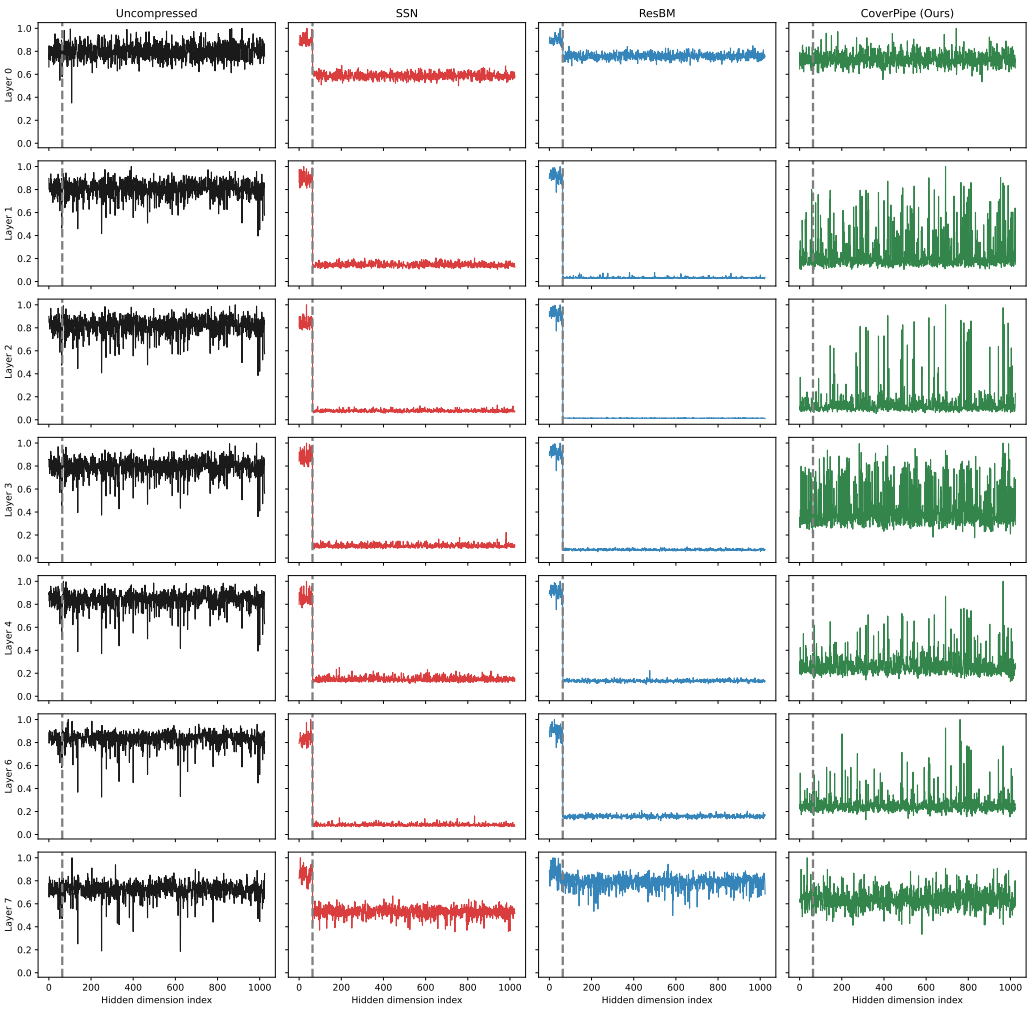


Figure 12: **Full results for the activation gradient magnitude across layers.** Each row is one layer other than the fifth stage shown in Figure 7; each column is one method. The dashed line marks the preserved-subspace boundary $h = 64$.

negligible gradient signal throughout the hidden space, while the compression baselines concentrate it in a narrow subspace.

D.2 More experiment results

Benefit of reducing index overhead A naive representation of index requires $\lceil \log_2 H \rceil$ bits per value. Given a communication budget of hn_b bits per input, this index cost can dominate the budget when H is large. We reduce this overhead with shared index set $\mathcal{G}$ and enumerative coding. Table 6 shows the effect at a fixed compression ratio $\rho = 64$: under the same communication budget, the number of activation values our scheme transmits per input grows faster than the naive scheme as H increases, from a $1.44\times$ gain at $H = 1024$ to a $1.54\times$ gain at $H = 8192$.

Table 6: Number of selected activation values per input under the same $64\times$ BF16 communication budget. Gain of ours ($\lambda = 0.6$) increases with the hidden dimension H.

H	h	Naive	Ours ($r + k_{\max}$)	Gain
1024	16	9	13 $(9 + 4)$	$1.44\times$
2048	32	18	27 $(19 + 8)$	$1.50\times$
4096	64	36	54 $(38 + 16)$	$1.50\times$
8192	128	70	108 $(76 + 32)$	$1.54\times$

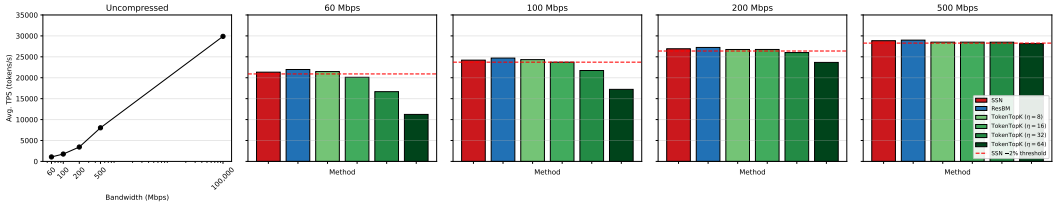


Figure 13: **Throughput across bandwidths: Dense.** Leftmost panel: Uncompressed TPS vs. bandwidth. Remaining panels: TPS by method at each evaluated bandwidth, with the dashed red line marking SSN's TPS minus 2%.

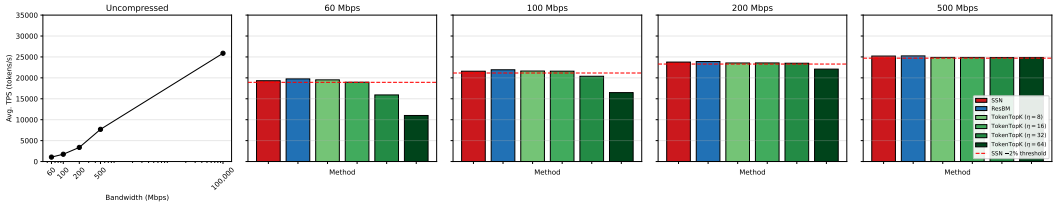


Figure 14: **Throughput across bandwidths: MoE.** Same layout as Figure 13.

Throughput across network bandwidths We provide throughput analysis of **C**over**P**ipe across bandwidth constraints and model architectures. Figure 13 and Figure 14 use the same configuration as Figure 3. In each figure, the leftmost panel shows Uncompressed throughput collapsing as bandwidth tightens, while the remaining four panels compare SSN, ResBM, and TokenTopK across the η sweep at each evaluated bandwidth, with the dashed red line marking the SSN -2% threshold used to select η in Appendix D.1. Because processing two experts per layer gives residual overlap a longer computation window in the MoE model, TokenTopK's bars clear the threshold at a smaller η in Figure 14 than at the same bandwidth in Figure 13, matching the $\eta_{\text{Dense}} \leq \eta_{\text{MoE}}$ pattern in Appendix D.1.

Baseline validation and evaluation in the prior-work configuration. We adopt the configuration of Aboudib et al. (2026) for two purposes: to validate our implementations of SSN and ResBM, and to test whether the advantage of **C**over**P**ipe extends to the shallow, wide model regime considered by prior work. This configuration differs from ours in shape and in token budget: a width-to-depth ratio of $H/L = 512$ against the $H/L = 128$ of our models (Table 4), and 2.5 tokens per parameter against the roughly 20 we use throughout (Appendix C.1).

We follow Table 1 of Aboudib et al. (2026): a Llama-3-style model with qk-norm, 8 blocks, $H = 4096$ (2.03B parameters), context length 1024, global batch 32 sequences, AdamW with weight decay 0.01, peak learning rate 2×10^{-4} with linear warmup and cosine decay, and 5B training tokens. A bottleneck is placed after every block but the last, giving seven compressed boundaries, at a compression ratio of 100 (bottleneck dimension 40). For the reproduction, we train five models: the uncompressed baseline with AdamW, and ResBM and SSN each with AdamW and with Muon (Jordan et al., 2024). We differ from that table in training on OpenWebText rather than C4, warming up over 600 steps rather than 500, decaying to 10% of the peak learning rate for both optimizers where the original decays to 1% for AdamW, and clipping gradients at 1.0, so absolute losses are not comparable with the published curves.

Figure 15(a) plots the validation loss of all five reproduced models. The ordering matches Figure 2(a) of Aboudib et al. (2026): ResBM is ahead of SSN under each optimizer, Muon is ahead of AdamW for both compressed architectures, and the uncompressed baseline is ahead of every compressed model. This agreement validates the implementations of SSN and ResBM used throughout the paper. Figure 15(b) evaluates **C**over**P**ipe in the same reproduced configuration and at the same main-path ratio $\rho = 100$, with $\eta \in \{0, 20, 40, 81\}$. Together, the two panels show that our baseline implementations reproduce the prior results and that **C**over**P**ipe achieves the lowest validation loss among the compression methods in the shallow, wide model regime considered by prior work.

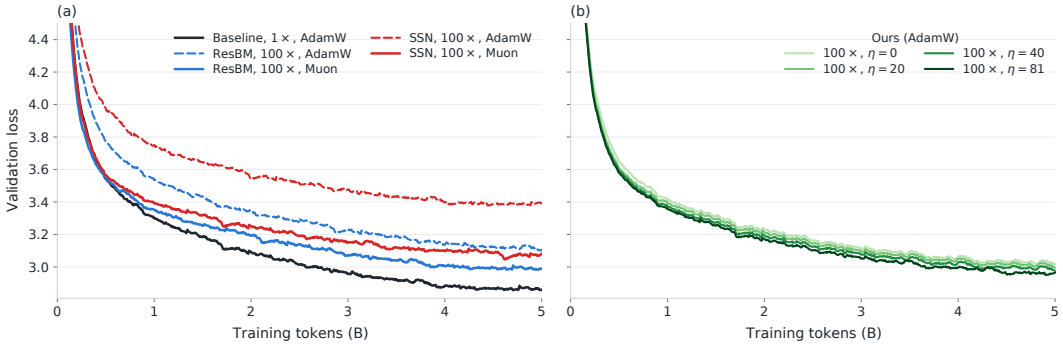


Figure 15: **Baseline reproduction and evaluation in a shallow, wide model regime.** Validation loss of the 2B, 8-block, $H = 4096$ model trained for 5B tokens, following the model configuration and token budget of Aboudib et al. (2026). **(a)** Our reproduction at a compression ratio of 100 matches the relative ordering reported in the prior work, validating our implementations of SSN and ResBM. **(b)** Under the same model configuration and main-path compression ratio $\rho = 100$, CoverPipe achieves the lowest validation loss among the compression methods; the evaluated η values are shown in the legend. Each curve is exponentially smoothed with decay 0.9.