Towards Network-Efficient Cross-Regional Inference via Learned Activation Compression

Regan McDonald
University of Michigan
Ann Arbor, Michigan, USA
mcregan@umich.edu

Marilyn Rego
University of Michigan
Ann Arbor, Michigan, USA
mrego@umich.edu

Ertza Warraich
Purdue University
West Lafayette, Indiana, USA
ewarraic@purdue.edu

Annus Zulfiqar
University of Michigan
Ann Arbor, Michigan, USA
zulfiqaa@umich.edu

Muhammad Shahbaz
University of Michigan
Ann Arbor, Michigan, USA
msbaz@umich.edu

Abstract

Large transformer models are increasingly deployed across geographically distributed GPU clusters due to capacity, cost, and locality constraints. When inference is partitioned across sites, intermediate activations must be transmitted over wide area network (WAN) links at each partition boundary, introducing significant communication overhead. We present Feather, a system that reduces this overhead by compressing intermediate activations before transmission and reconstructing them before downstream layers resume execution.

Feather learns a compact representation of activation tensors using a lightweight neural codec trained with a reconstruction objective while keeping the original model frozen. Across encoder and decoder transformer models, Feather achieves up to 48× activation compression while maintaining accuracy close to the baseline model. Under representative WAN conditions (e.g., 10 Gbps bandwidth and a 10 ms RTT), this reduction yields up to 4.96× improvement in end-to-end latency, consistently outperforming existing compression schemes, including linear autoencoder, PCA, and SVD.

Keywords

Distributed LLM Inference; Cross-Region Inference; Activation Compression; Communication-Efficient AI; Model Partitioning; Wide-Area Networks

ACM Reference Format:
Regan McDonald, Marilyn Rego, Ertza Warraich, Annus Zulfiqar, and Muhammad Shahbaz. 2026. Towards Network-Efficient Cross-Regional Inference via Learned Activation Compression. In *Workshop on Networks for AI Computing (NAIC '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3789240.3828744

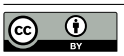
1 Introduction

Modern inference workloads span multiple machines, clusters, and even data centers. Frontier transformer models with hundreds of billions of parameters exceed the memory capacity of a single GPU node, requiring inference to be partitioned across multiple devices and machines [31, 35]. At the same time, production serving systems must maintain availability and latency targets under unpredictable demand, routing requests across geographic regions to absorb traffic spikes, satisfy data residency constraints, or compensate for localized capacity limits. For example, Amazon Bedrock's Cross-Region Inference Service dynamically routes inference requests across multiple regions to sustain low-latency responses under variable load [2]. The infrastructure supporting such deployments—including model partitioning, WAN connectivity, and distributed serving pipelines—is already widely used in production systems.

When such partitioned inference spans geographically separated sites, intermediate activations must be transmitted across wide area network (WAN) links at each partition boundary [4, 15, 22]. Unlike model weights, which are transferred infrequently during deployment or updates, activations are generated for every request [22, 36]. Their volume, therefore, scales directly with the service's request rate. In large-scale deployments serving millions of requests per day, activation transfer becomes a persistent and substantial communication cost across the inference pipeline [6, 10, 20, 21, 34].

WANs further amplify this challenge. Cross-regional datacenter links provide lower throughput, higher latency, and greater variability than intra-datacenter fabrics [11, 14, 28, 29]. The bandwidth gap between intra-rack communication and trans-regional WAN links can be substantial, making activation transfers that are inexpensive within a datacenter significantly more costly across regions. In addition, large activation transfers compete with other traffic on shared inter-datacenter links, potentially increasing congestion and latency. Finally, cloud egress charges introduce an additional economic cost to these transfers. Together, these factors make WAN communication a key constraint in cross-region inference deployments. These constraints raise a natural question: *how can the communication cost of activation transfer be reduced?*

Existing systems primarily focus on optimizing where computation runs, rather than reducing the amount of data transferred between stages. Model parallelism and pipeline execution frameworks distribute large models across GPUs and machines so that models exceeding single-device memory can still run [4, 22, 31, 32], but they introduce partition boundaries where the full activation tensor must cross the network unchanged. Routing frameworks



NAIC '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3828744

optimize request placement across regions but treat activation volume as fixed [2]. Prior work on edge-cloud partitioning and inter-datacenter scheduling similarly focuses on optimizing the placement of split points rather than reducing the bandwidth cost at those points. As a result, the volume of activation traffic remains largely unchanged.

These observations motivate Feather, a learned activation-compression framework for split-transformer inference. Rather than applying compression independently of the model, Feather trains lightweight encoder-decoder adapters directly at the model's partition boundary to reconstruct intermediate hidden states with the fidelity required by frozen downstream layers. Because the downstream layers remain fixed and cannot be retrained to compensate for distorted inputs, the adapter must learn to reconstruct the activations rather than merely compress them. The key idea is to learn compression with respect to the split itself: *the adapters are supervised using reconstruction loss at the partition boundary, ensuring that the compressed representation preserves the structure expected by the downstream model.* Feather operates at the partition boundary between model stages and can be applied regardless of where the model is split across devices or datacenters. Feather enables up to 48× reduction in activation bandwidth while remaining within 2% of the uncompressed model's accuracy, yielding end-to-end speedups of up to 4.96× under representative inter-datacenter network conditions.

Our contributions are as follows:

- We propose Feather, a learned activation compression system for split transformer inference that reduces the size of intermediate activations transmitted between model partitions.
- We show that learned activation compression significantly outperforms classical dimensionality reduction techniques (e.g., linear autoencoder, PCA, and SVD) in preserving inference accuracy, achieving up to 30% higher accuracy at high compression ratios across both encoder and decoder transformer architectures.
- We show that reducing activation size directly translates into system-level benefits, achieving up to 48× reduction in activation bandwidth and 4.96× end-to-end speedup under representative cross-regional network conditions.

2 Background and Related Work

Distributed Inference and Model Partitioning. As transformer models grow beyond the memory capacity of a single device, inference is increasingly partitioned across multiple GPUs, machines, and datacenters using pipeline and tensor parallelism [31, 32]. This creates partition boundaries where intermediate activations must be transferred between stages. Prior work has studied how to place these boundaries efficiently. Kang et al. [19] optimize edge-cloud split points to balance latency and energy consumption, while Jiang et al. [16] study multi-datacenter deployments in which inter-datacenter bandwidth is the primary constraint. Production systems, e.g., Amazon Bedrock's Cross-Region Inference Service, route requests across regions to maintain performance under variable demand [2]. These works focus on optimizing where models are partitioned or how requests are routed, but they treat the volume of data transmitted at partition boundaries as fixed. Feather instead targets the bandwidth required to transfer activations across these boundaries.

Communication Compression in Distributed Training. Reducing communication overhead has been extensively studied in distributed training. Techniques such as gradient quantization, sparsification, and low-rank approximation reduce the volume of data exchanged during gradient aggregation [1, 23, 37]. These approaches are effective because training is iterative: compression errors introduced in one step can be corrected in subsequent updates. Inference does not share this property. A distortion introduced at a partition boundary propagates through the remaining frozen layers of the model and directly affects the final prediction.

Classical Dimensionality Reduction. Classical dimensionality reduction methods provide natural baselines for compressing high-dimensional representations. Principal component analysis (PCA) [18, 27] and truncated singular value decomposition (SVD) [8, 9] project activations into a lower-dimensional linear subspace learned from collected samples. However, these projections are optimized for variance preservation or matrix approximation rather than for reconstructing activations required by downstream layers, and they rely on a fixed projection basis that cannot adapt to shifts in activation distributions during deployment. They reduce activation dimensionality, oblivious to how the resulting representations will be used by downstream transformer layers, and often introduce distortions that impact model accuracy.

Activation Compression for Split Inference. Several works compress intermediate representations at model split points for distributed inference. Matsubara et al. [26] learn supervised compression modules that jointly retrain downstream layers with the compressor. SLICER [36] applies sparsification and adaptive quantization at the split point without retraining the model. However, these methods do not explicitly optimize for reconstructing the activation structure expected by downstream transformer layers, which can lead to accuracy degradation at higher compression ratios. In contrast, Feather learns lightweight compression-reconstruction adapters directly at the partition boundary using reconstruction supervision while keeping all model parameters frozen. Our design is based on adaptors [13], repurposing lightweight trainable modules within a frozen model for communication-efficient activation transfer.

3 Feather Design

Figure 1 demonstrates Feather, our learned neural codec that reduces the communication overhead of cross-regional distributed inference by compressing intermediate activations before they are transmitted across the network.[1] Feather inserts lightweight Feather$^{\downarrow}$ and Feather$^{\uparrow}$ adaptor modules at the partition boundary, allowing large activation tensors to be replaced with compact representations during WAN transmission while preserving the structure expected by downstream transformer layers.

Overview. Feather preserves the structural properties of the activations expected by downstream layers to prevent compression errors at the partition boundary from propagating through the

[1]We focus on transformer-based architectures, but the ideas apply broadly to modern AI models based on neural networks (NNs).

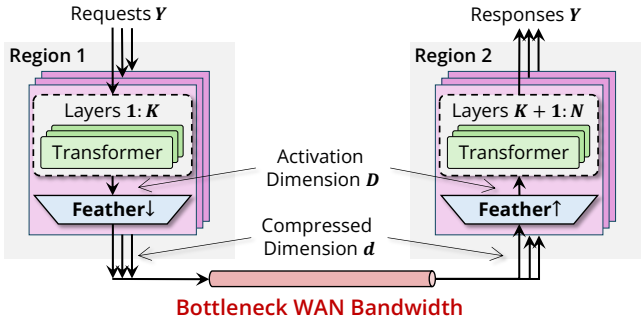


Figure 1: Feather overview: the model is partitioned across two regions: K layers in Region 1 and $K+1:N$ in Region 2. For Y parallel prompt requests, Feather$^{\downarrow}$ compresses their intermediate activations to dimension d before transmitting over the bottleneck wide-area link. On the receiver side, Feather$^{\uparrow}$ reconstructs the activations to original dimension D before inference continues through remaining transformer blocks.

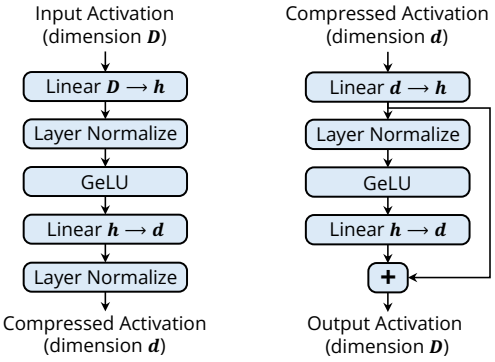
remainder of the model. To that end, Feather learns *split-aware compression* mappings specifically for the split inference setting using lightweight neural codec adaptors placed at the partition boundary. As shown in Figure 1, Feather$^{\downarrow}$ executes on the upstream model partition and Feather$^{\uparrow}$ reconstructs activations for downstream layers.

In transformer models, the communication cost of split inference is dominated by the activation tensor produced at the partition boundary, with dimensions $Y \times T \times D$ where Y is the number of prompts (inference requests) being processed in parallel, T is the sequence length (#tokens), and D is the hidden dimension of the transformer layer at the split. Feather reduces this communication cost by introducing a *bottleneck activation representation* that projects each token vector from dimension D to a lower dimension d, producing a compressed tensor of size $Y \times T \times d$, yielding a compression ratio of D/d. Moreover, to preserve the token-level semantics expected by downstream attention layers, compression is applied independently to each token representation rather than across tokens, ensuring that the sequence structure of the activation tensor is maintained during efficient parallel execution on GPUs.

3.1 Feather Adaptor Architecture

We now describe the architecture of the Feather adaptors, Feather$^{\downarrow}$ and Feather$^{\uparrow}$. As shown in Figure 2, both adaptors use lightweight multilayer perceptrons (MLPs) to implement the compression transformation. Rather than relying on fixed linear projections, the nonlinear structure of these MLPs allows the adaptors to learn task-relevant compression and reconstruction mappings that better preserve the hidden-state structure expected by downstream transformer layers (§4).

Nonlinear Projection. Both Feather$^{\downarrow}$ (compression adaptor) and Feather$^{\uparrow}$ (decompression adaptor) are implemented as lightweight nonlinear MLPs that transform token activations around a bottleneck dimension d. The Feather$^{\downarrow}$ adaptor (Figure 2a) maps each token activation from the transformer hidden dimension D to the bottleneck representation d, while the Feather$^{\uparrow}$ adaptor (Figure 2b) reconstructs the activation by expanding the compressed representation back to dimension D. Using nonlinear MLPs enables the



(a) Feather↓ Adaptor **(b) Feather↑ Adaptor**

Figure 2: Feather adaptors: Feather$^{\uparrow}$ (compressor) and Feather$^{\downarrow}$ (decompressor). Both modules use a nonlinear MLP with an intermediate dimension h. In tandem, they support a compression ratio of D/d.

adaptors to capture correlations among hidden dimensions and preserve the information required by downstream transformer layers, which is particularly important when compressing activations under aggressive compression ratios.

In the compression path, each token vector is projected from D to an intermediate dimension h, followed by layer normalization and a GELU nonlinearity, and then projected to the bottleneck dimension d. Along the decompression path, the compressed representation is projected from d to an intermediate dimension h and then expanded back to the original hidden dimension D, thus reconstructing the activation tensor expected by the downstream transformer layers.

Adaptive Bottleneck Scaling. To maintain sufficient transformation capacity under varying compression ratios, the intermediate dimension h is scaled relative to the bottleneck dimension d. Both the Feather$^{\downarrow}$ and Feather$^{\uparrow}$ adaptors use the same intermediate dimension so that the compressor and decompressor maintain symmetric expressive capacity. Specifically, the hidden dimension is set to $h = 32d$ when $d < 8$ and $h = 64d$ when $d \geq 8$. This adaptive scaling increases the expressive power of the compression and reconstruction transformations while keeping the computational overhead small compared to the transformer layers.

Residual Reconstruction Pathway. Reconstructing activations from a highly compressed representation can introduce reconstruction errors that propagate through downstream transformer layers. To improve reconstruction stability, the Feather$^{\uparrow}$ adaptor incorporates a residual reconstruction pathway, as shown in Figure 2b, that introduces a skip connection from the intermediate representation to the final output layer. This residual block allows the decompression adaptor to learn corrective refinements on top of a coarse reconstruction produced from the bottleneck representation. By enabling the adaptor to focus on modeling residual reconstruction errors, this design improves the fidelity of reconstructed activations while maintaining a lightweight architecture suitable for execution at the split inference boundary.

3.2 Training Feather Adaptors

We train the Feather$^{\downarrow}$ and Feather$^{\uparrow}$ adaptors to encode transformer activations into a compact bottleneck representation while

preserving the information required by downstream transformer layers. To achieve this, we freeze the parameters of the original transformer and train only the Feather adaptors, following prior adaptor-based architectures [13]. This design allows the compression mechanism to be trained independently without modifying the pre-trained model weights.

During training, we capture the activations at the partition boundary, denoted by $H \in \mathbb{R}^{B\times T\times D}$, where B is the batch size, T is the sequence length (#tokens), and D is the hidden dimension of the transformer layer. The Feather$^{\downarrow}$ adaptor compresses these activations to produce a bottleneck representation $H^{\downarrow} \in \mathbb{R}^{B\times T\times d}$, and the Feather$^{\uparrow}$ adaptor reconstructs the activations as $H^{\uparrow} \in \mathbb{R}^{B\times T\times D}$. Because the training objective is to reconstruct the original activations, the input activations H serve as the supervision signal for the Feather$^{\uparrow}$ adaptor, effectively acting as the training labels for the reconstruction process. Training, therefore, optimizes the adaptors so that the reconstructed activations $H^{\uparrow}$ closely match the original activations produced by the transformer.

Reconstruction Objective. The training objective minimizes the difference between the original activations H and the reconstructed activations $H^{\uparrow}$ using a combination of mean-squared error and cosine similarity, following prior work on representation matching and activation reconstruction [12, 17, 33]. This reconstruction objective function is given as follows:

$$\mathcal{L} = \frac{\sum_{i,t} m_{it}\|H_{it} - H^{\uparrow}_{it}\|^2}{\sum_{i,t} m_{it}} + 1 - \frac{\sum_{i,t} m_{it}\cos(H_{it}, H^{\uparrow}_{it})}{\sum_{i,t} m_{it}}$$

Here m_{it} denotes the attention mask for token t in sequence i. The mask excludes padding tokens so that the adaptors are trained only on tokens that participate in transformer computation.

3.3 Feather Analysis

Compressing the activations lowers communication cost but introduces additional computation for compression and reconstruction. Whether compression improves end-to-end latency, therefore, depends on the tradeoff between reduced transmission time and the added compute overhead.

Let W denote the available WAN bandwidth and RTT denote the round-trip latency of the link. When Y inference requests are processed concurrently and equally share the available bandwidth, the effective bandwidth available to each request becomes W/Y. Without compression, transmitting the activation tensor of size $T \cdot D$ incurs a latency of $t = (T \cdot D)/(W/Y) + \text{RTT}/2 = (Y \cdot T \cdot D)/W + \text{RTT}/2$.

With Feather, the transmitted payload is reduced to $T \cdot d$ values. Let $t^{\downarrow}$ and $t^{\uparrow}$ denote the execution time of the Feather$^{\downarrow}$ and Feather$^{\uparrow}$ adaptors, respectively. The resulting latency becomes $t^{\downarrow\uparrow} = (T\cdot d)/(W/Y)+\text{RTT}/2+t^{\downarrow}+t^{\uparrow} = (Y\cdot T\cdot d)/W+\text{RTT}/2+t^{\downarrow}+t^{\uparrow}$.

Compression reduces end-to-end latency when $t^{\downarrow\uparrow} < t_{\text{base}}$, which yields the condition $t^{\downarrow} + t^{\uparrow} < (Y \cdot T \cdot (D - d))/W$. Intuitively, compression is beneficial when the additional compute time required by the adaptors is smaller than the network time saved by transmitting compressed activations.

This condition becomes easier to satisfy when WAN bandwidth is limited, when the compression ratio D/d is large, when many prompts share the same link (Y is large), or when the transformer hidden dimension D is large, as is common in modern large-scale transformer models.

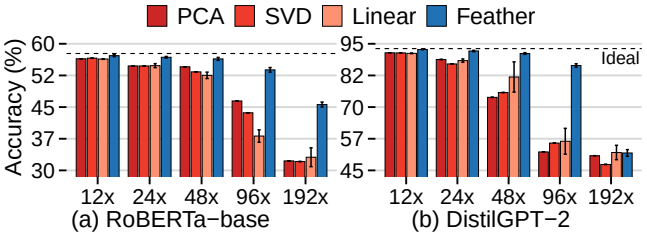


Figure 3: End-to-end test accuracy of Feather and baselines across compression ratios for encoder (RoBERTa-base) and decoder (DistilGPT-2) models.

4 Preliminary Evaluations

We evaluate Feather's end-to-end performance in preserving model test accuracy for the downstream task under activation compression, and its system-level impact on inference latency under realistic WAN conditions. We also benchmark Feather's reconstruction fidelity measured using signal-to-noise ratio (SNR) defined as $10\log_{10}\left(\frac{\text{signal power}}{\text{noise power}}\right)$, independently of downstream task performance. For parametric methods (Feather and the linear autoencoder), accuracy and SNR are reported as mean ± standard deviation across five independent random seeds (42, 123, 456, 789, 2024).

4.1 Experimental Setup

Test Environments. We simulate deployments where inference is partitioned across two devices connected through a WAN. In all experiments, the transformer model is split at its architectural midpoint so that early layers execute on the first device and later layers execute on the second device. The compressed activation tensor produced at the partition boundary is transmitted between the two devices and reconstructed before continuing the forward pass. We evaluate network conditions using a 10 Gbps link, consistent with the capacity of dedicated inter-datacenter connections offered by major cloud providers [3, 5]. Round-trip latencies of 10 ms, 25 ms, and 50 ms are used to represent typical cross-region and inter-datacenter WAN deployments. Feather$^{\downarrow}$ and Feather$^{\uparrow}$ compute latency ($t^{\downarrow}$ and $t^{\uparrow}$) are measured using CUDA events and averaged over 100 timed iterations with 20 warmup iterations on batches of 64 real tokens. These measurements are combined with network transfer time to evaluate end-to-end latency (§3.3).

Baselines, Workloads, and Parameter Settings. We evaluate Feather on two transformer architectures spanning both encoder/decoder model families. For the encoder setting, we use RoBERTa-base [24] fine-tuned on GoEmotions [7], a 28-class emotion classification benchmark derived from Reddit comments. For the decoder setting, we fine-tune DistilGPT-2 [30] on the IMDB sentiment classification dataset [25], a widely used binary sentiment benchmark consisting of movie reviews labeled as positive or negative. Both models have a hidden dimension $D = 768$, and all baseline model parameters remain frozen throughout adapter training and inference. Padding positions are excluded from both the forward pass and gradient updates using attention masks.

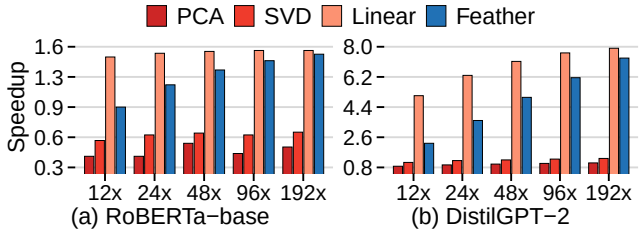


Figure 4: End-to-end speedup for activation compression, transmission, and decompression against compression ratios at RTT = 10 *ms*. Speedup is relative to transmitting uncompressed activations (baseline RoBERTa-base: 7.9 *ms*, DistilGPT-2: 42.1 *ms*). Results for RTT = 25 *ms* and 50 *ms* follow the same trend.

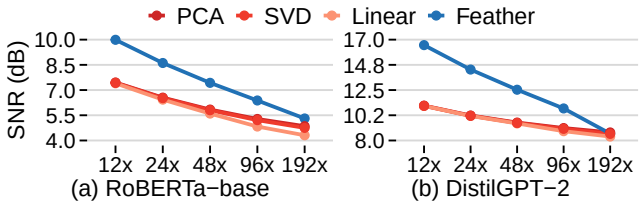


Figure 5: Reconstruction quality as SNR (in dB) of Feather and baselines across compression ratios for encoder (RoBERTa-base) and decoder (DistilGPT-2) models. Higher SNR indicates better fidelity to the original activations.

We evaluate activation compression across compressed dimensions $d \in \{64, 32, 16, 18, 4\}$, corresponding to compression ratios ranging from 12× to 192×. Across these settings, we compare Feather against three baselines. The first baseline is a linear autoencoder consisting of a single linear encoder and decoder without nonlinearities, isolating the contribution of the nonlinear bottleneck used in Feather. The second and third baselines are PCA and truncated SVD projections fitted directly on training-set activations. These methods represent static linear projections that do not incorporate a learned reconstruction objective.

The compression adapter is trained using the reconstruction objective described in §3.2, with all backbone model parameters frozen. Training is performed for 5 epochs using the Adam optimizer with a learning rate of 1×10^{-3} and a cosine annealing schedule. The adapter is trained once per compression ratio and architecture with the same dataset used to fine-tune the baseline model per task, and no online adaptation is performed during inference.

4.2 End-to-End Model Accuracy

Figure 3a shows downstream task accuracy for our encoder-only model, RoBERTa-base. Feather consistently outperforms all baselines, with the gap widening under aggressive compression. At 48× ($d = 16$), Feather retains 97.8% of baseline performance (56.42%), versus 52.52% for Linear and 54.55% for PCA. At 96× ($d = 8$), Feather achieves 53.82%, substantially ahead of Linear (38.14%), PCA (46.45%), and SVD (43.65%). PCA and SVD, being static decompositions, cannot adapt to the task-relevant structure of hidden states and degrade most sharply under aggressive compression. The learned linear autoencoder recovers some of this structure through training, but its lack of nonlinearity limits its ability to preserve the representations required by downstream transformer layers.

For the decoder-only model, DistilGPT-2, Figure 3b shows that Feather's learned compression generalizes well to decoder architectures. Feather again outperforms all baselines, achieving 86.43% at 96× compression versus 56.51% for Linear, while PCA and SVD degrade substantially at this level (52.30% and 55.80%). At the extreme compression ratio of 192× ($d = 4$), all methods converge to similar performance levels, suggesting that severe dimensionality reduction leaves little room for any technique to recover lost activations. The performance gap is larger than in the encoder (29.92% vs. 15.68%, Figure 3a), consistent with causally contextualized hidden states being less globally structured and therefore harder to compress with static linear projections.

4.3 End-to-End Network Speedup

We evaluate the system-level impact of activation compression on end-to-end inference latency based on the analysis described in §3.3. Figure 4a reports speedup results for RoBERTa-base across compression ratios from 12× to 192× at 10 Gbps and an RTT of 10 ms. PCA and SVD, running on CPU, consistently yield slowdowns (0.42×–0.52× and 0.59×–0.68×, respectively), making them unsuitable for latency-sensitive deployments. Linear achieves positive speedups throughout (1.49× at 12×, plateauing at 1.56×), owing to its negligible compute overhead. Feather starts below Linear at low compression (0.95× at 12×) due to its nonlinear adapter overhead, but converges at high compression (1.52× at 192×) while maintaining substantially higher accuracy (45.60% vs. 33.12%, Figure 3).

For DistilGPT-2 (Figure 4b), the same trend holds but with larger speedups due to higher per-layer transfer volume. Linear reaches 7.91× at 192× compression but collapses in accuracy (52.07%), while Feather achieves 6.14× at 96× and retains 86.43% accuracy—a gap of nearly 30 percentage points over Linear. Overall, PCA and SVD are not viable for network acceleration, Linear trades accuracy for throughput, and Feather provides the best accuracy–speedup tradeoff in bandwidth-constrained WAN deployments.

4.4 Reconstruction Fidelity

While downstream accuracy reflects task-level performance, it does not directly measure how well compressed activations reconstruct the original hidden representations. We therefore evaluate reconstruction quality using signal-to-noise ratio (SNR), where higher SNR indicates closer fidelity to the upstream model's hidden states.

Figure 5a reports SNR across compression ratios for RoBERTa-base. Feather consistently achieves higher SNR than all baselines, with degradation remaining gradual as compression increases—from 9.99 dB at 12× down to 5.31 dB at 192×. At 48× compression, Feather achieves 7.43 dB versus 5.60 dB for Linear; at 96×, the gap persists at 6.38 dB versus 4.83 dB. Linear, PCA, and SVD degrade more sharply, consistent with the accuracy collapse observed at high compression ratios, as shown in Figure 3.

Figure 5b shows a similar pattern for DistilGPT-2, with overall higher SNR values reflecting the decoder's larger hidden dimension. Feather decreases gradually from 16.51 dB at 12× to 8.61 dB at 192×, while at 96× it retains 10.86 dB versus 8.83 dB for Linear. The baselines again degrade more sharply, indicating that the nonlinear

adapter better preserves hidden-state structure, particularly in the more challenging causal decoder setting.

5 Discussion & Call to Arms

Compression has become a central technique in modern machine learning systems. Quantization, pruning, and distillation are widely used to reduce the compute and memory footprint of large models. These techniques primarily target efficient model storage and computation on accelerators. As inference increasingly spans machines and datacenters, however, another resource becomes equally important: *network bandwidth*. Intermediate activations must be transmitted between model partitions, and the latency of these transfers directly determines when downstream computation can proceed. In distributed inference pipelines, GPUs frequently stall while waiting for activations to arrive, making the communication channel itself a critical system bottleneck.

While prior work has explored compression for model parameters and training communication, activation transmission during split inference remains relatively underexplored. In this work, we take inspiration from model-compression techniques and apply similar ideas to the communication channel between model partitions. By learning lightweight adapters that compress intermediate activations, Feather reduces the volume of data transmitted across WAN links while preserving the hidden-state structure expected by downstream transformer layers.

This perspective suggests a broader research direction. As inference systems become increasingly distributed, compression mechanisms should be integrated directly into the inference pipeline rather than treated as an external optimization. This includes co-designing activation compression with model training, combining dimensionality reduction with quantization, and developing system abstractions that explicitly account for communication costs.

Towards Communication-Aware Inference Systems. More broadly, we believe that network communication will play an increasingly important role in the design of large-scale machine learning systems. Just as model compression reshaped how the community thinks about compute efficiency, communication-aware architectures may reshape how we design distributed inference pipelines.

The infrastructure for distributed inference already exists—models are partitioned across machines, serving pipelines span datacenters, and WAN links connect them. What remains largely unexplored is how to design inference systems that treat the movement of activations across these links as a first-class systems problem. We believe that making communication-aware inference a core design principle is an important next step for large-scale machine learning systems, and we hope this work encourages further exploration in that direction.

6 Ethical Considerations

This work does not involve human subjects, personal data, or sensitive user information. The experiments are conducted on publicly available datasets and synthetic or simulated environments.

References

[1] Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *NeurIPS* (2017).

[2] Amazon Web Services. Amazon Bedrock Cross-Region Inference. https://docs.aws.amazon.com/bedrock/latest/userguide/cross-region-inference.html. Accessed: Feb. 2026.

[3] Amazon Web Services. AWS Direct Connect. https://docs.aws.amazon.com/directconnect/latest/UserGuide/Welcome.html. Accessed: Feb. 2026.

[4] Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2022).

[5] Azure. Azure ExpressRoute. https://azure.microsoft.com/en-us/products/expressroute. Accessed: Feb. 2026.

[6] Chen, G., Lv, J., Ye, K., Gu, T., and Xu, C. Scalable and Fast Inference Serving via Hybrid Communication Scheduling on Heterogeneous Networks. In *2025 IEEE International Conference on Cluster Computing (CLUSTER)* (2025).

[7] Demszky, D., Movshovitz-Attias, Y., Ko, M., Munkhdalai, T., Peng, N., McClosky, D., and Ruder, S. GoEmotions: A Dataset of Fine-Grained Emotions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020).

[8] Eckart, C., and Young, G. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika* (1936).

[9] Halko, N., Martinsson, P.-G., and Tropp, J. A. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review* (2011).

[10] Haziza, D., Chou, T., Choudhary, D., Wehrstedt, L., Massa, F., Yu, J., Jeong, G., Rao, S., Labatut, P., and Cai, J. Accelerating Transformer Inference and Training with 2: 4 Activation Sparsity. *arXiv preprint arXiv:2503.16672* (2025).

[11] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM* (2013).

[12] Hou, Z., Liu, X., Cen, Y., Dong, Y., Yang, H., Wang, C., and Tang, J. GraphMAE: Self-Supervised Masked Graph Autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2022).

[13] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-Efficient Transfer Learning for NLP. In *ACM ICML* (2019).

[14] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al. B4: Experience with a Globally-Deployed Software Defined WAN. *ACM SIGCOMM CCR* (2013).

[15] Jiang, Y., Yan, R., Yao, X., Zhou, Y., Chen, B., and Yuan, B. Hexgen: Generative Inference of Large Language Model over Heterogeneous Environment. *arXiv preprint arXiv:2311.11514* (2023).

[16] Jiang, Z., Cao, Z., Zhou, S., Zhu, Y., Chen, K., and Shu, Y. Joint Optimization of Traffic Engineering and Resource Allocation in Inter-Datacenter Networks. In *ACM SIGCOMM* (2018).

[17] Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (2020).

[18] Jolliffe, I. T. *Principal Component Analysis*, 2nd ed. Springer, 2002.

[19] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *ACM ASPLOS* (2017).

[20] Li, Q., Zhang, B., Kang, H., Xu, T., Qian, Y., Xie, Y., and Ma, L. FlashCommunication V2: Bit Splitting and Spike Reserving for Any Bit Communication. *arXiv preprint arXiv:2508.03760* (2025).

[21] Li, Q., Zhang, B., Ye, L., Zhang, Y., Wu, W., Sun, Y., Ma, L., and Xie, Y. Flash Communication: Reducing Tensor Parallelization Bottleneck for Fast Large Language Model Inference. *arXiv preprint arXiv:2412.04964* (2024).

[22] Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., et al. {AlpaServe}: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *USENIX OSDI* (2023).

[23] Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *ICLR* (2018).

[24] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* (2019).

[25] Maas, A., Daly, R., Pham, P., Huang, D., Ng, A., and Potts, C. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics* (2011).

[26] Matsubara, Y., Yang, R., Levorato, M., and Mandt, S. Supervised Compression for Resource-Constrained Edge Computing Systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2022).

[27] Pearson, K. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine* (1901).

[28] Persico, V., Botta, A., Marchetta, P., Montieri, A., and Pescapé, A. On the Performance of the Wide-Area Networks Interconnecting Public-Cloud Datacenters Around the Globe. *Computer Networks* (2017).

[29] Persico, V., D'Antonio, S., Botta, S., and Pescapé, A. A First Look at Public-Cloud Inter-Datacenter Network Performance. In *Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM)* (Washington, DC, USA, 2016).

[30] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language Models Are Unsupervised Multitask Learners. *OpenAI blog* (2019).

[31] Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2020).

[32] Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. https://www.deepspeed.ai/, 2020.

[33] Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Bacon, P.-L., and Courville, A. FitNets: Hints for Thin Deep Nets. *arXiv preprint arXiv:1412.6550* (2014).

[34] Shao, J., Mao, Y., and Zhang, J. Learning Task-Oriented Communication for Edge Inference: An Information Bottleneck Approach. *IEEE Journal on Selected Areas in Communications* (2021).

[35] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[36] Sung, M., Im, S., Bang, D., Kim, I.-M., Yun, S., and Kang, J.-M. Why Should the Server Do It All?: A Scalable, Versatile, and Model-Agnostic Framework for Server-Light DNN Inference over Massively Distributed Clients via Training-Free Intermediate Feature Compression. https://arxiv.org/abs/2511.11608, 2025. arXiv preprint.

[37] Vogels, T., Karimireddy, S. P., and Jaggi, M. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. In *NeurIPS* (2019).