CINN: A COMMUNICATION-INFORMED NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

As models grow, distributed pre-training must repeatedly transfer model weights and parameter fragments across devices or data centers, so the data volume increases and transfer time lengthens. The next compute step then waits for the data to arrive, leaving the computing devices idle. To address this problem, this paper proposes a Communication-Informed Neural Network (CINN), which improves communication efficiency by reducing the dimensionality of the model weights and parameter fragments to be transmitted. Unlike existing dimensionality-reduction methods that mainly target numerical reconstruction or rely on fixed linear maps, CINN reshapes the local geometry of model parameter fragments through an input-dependent nonlinear transformation followed by a compressive projection, with both components adapted to the task. Jacobian–gradient response strengthens the sensitivity of the low-dimensional representation to task-relevant directions and reduces information loss under the compressive projection, while an orthogonality regularizer reduces directional overlap among transmitted coordinates so that limited communication resources preferentially retain task-relevant information. In CIFAR-10 distributed training experiments with ResNet-56, CINN compresses the accumulated outer-loop parameter difference after every 10 local optimization steps, mapping each 1028-dimensional parameter block to a low-dimensional latent plus one scale factor. At $8\times$ compression, CINN is only 4.72 points below uncompressed transmission. When each block is compressed to 16 dimensions, CINN reaches 72.78%, 9.98 points above linear projection and 17.46 points above PCA. Under a 200 ms / 1 Gbps wide-area network (WAN) model, CINN at $1/8$ compression reduces the mean epoch time of a 1B-parameter FP16 payload from 346.1 s to 62.6 s.

Keywords: distributed model training, communication bottleneck, training-state compression

1 INTRODUCTION

Frontier models can no longer be trained on a single accelerator and must be trained across many devices in a compute cluster. Google's Decoupled DiLoCo (Douillard et al., 2023) notes that conventional Single Program, Multiple Data (SPMD) requires every accelerator to maintain a globally consistent copy of the model weights and synchronize at every step. As models grow, the volume of parameters transferred across devices and datacenters increases. For a 12-billion-parameter dense model, the full FP16 weights occupy about 24 GB. Consequently, communication can consume a large fraction of training time, since GPUs must wait for model weights and parameter fragments to arrive before computation resumes.

Decoupled DiLoCo reduces synchronization frequency by allowing each compute node to perform multiple local updates before sending parameter fragments to a central synchronizer. Although this approach overlaps communication with longer local computation, it does not reduce the amount of data transmitted during each synchronization. After fragmentation, each transmission may still contain a parameter block with billions of dimensions, and sending such a block over a 2–5 Gbps link can take seconds to tens of seconds. These high-dimensional parameter blocks therefore still contain redundancy and task-irrelevant components that should be further compressed within a single transfer.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Compressing the model weights and parameter fragments is a direct way to reduce communication overhead. Lowering the transmission dimension or the number of effective transmitted coordinates reduces the communication payload, while reducing redundancy improves the utilization of the available communication budget. However, mapping a high-dimensional state to a low-dimensional representation necessarily causes information loss. The key question is therefore which information should be retained under a fixed communication budget. Principal component analysis tends to preserve directions with large variance or energy, but may discard low-magnitude directions that are important to the task. A fixed linear projection uses the same low-dimensional subspace for all inputs and may place task-relevant directions in its invisible subspace.

The proposed Communication-Informed Neural Network (CINN) first applies an input-dependent nonlinear transformation to the model weights and parameter fragments, reorganizing the local geometry of the transmitted state, and then produces a compact representation through low-dimensional projection. The compressor is trained jointly with the downstream task so that the compact representation responds preferentially to directions that affect the task loss. CINN also introduces a Jacobian–gradient response to encourage small variations along loss-sensitive directions to produce observable changes in the transmitted code. In addition, an orthogonality regularizer reduces repeated representation of the same direction by different transmitted coordinates, allowing limited communication resources to retain more task-relevant information.

In the experiments, CINN partitions each DiLoCo outer-loop update into blocks of 1,028 coordinates and compresses every block into a 128-dimensional latent representation plus one scale factor after every 10 local optimization steps. At $8\times$ compression after 200 epochs, CINN is only 4.72 points below uncompressed transmission, while reducing the transmitted payload to approximately $1/8$ of the original. CINN also remains above PCA and supervised linear projection under the same bottleneck. Under a simulated 200 ms / 1 Gbps wide-area network (WAN) model, a 1B-parameter FP16 payload requires 346.1 s per epoch with direct transmission. CINN reduces this time to 62.6 s, 42.2 s, and 31.9 s at compression ratios of $1/8$, $1/16$, and $1/32$, respectively, substantially accelerating the outer loop.

The main contributions of this paper are as follows:

1. A proof that linear compression necessarily has a null space, and that any nonzero component of the task gradient in that subspace can change the loss to first order without changing the transmitted code. CINN is further shown to reorient this null space with an input-dependent nonlinear map, so that currently important directions can leave the invisible subspace, together with the design of a Jacobian–gradient response and an orthogonality regularizer.
2. An input-dependent nonlinear CINN that learns compact representations through the task loss, Jacobian–gradient response, and an orthogonality regularizer, under a prescribed communication budget.
3. Experimental results showing that CINN retains a usable task update under strong compression, outperforms PCA and linear projection at the same bottleneck, and shortens outer-loop communication on bandwidth-limited links.

2 Related Work

2.1 Communication Bottlenecks in Distributed Training

Distributed pre-training coordinates model-weight and parameter-fragment updates among worker nodes through a central node or sharded servers, whereas data-parallel systems commonly use AllReduce to aggregate gradients (Dean et al., 2012; Li et al., 2014; Sergeev & Del Balso, 2018; Goyal et al., 2017). Existing systems mainly shorten communication time in the following ways: low-bit quantization uses fewer bits to represent gradients or parameters, Top-k sparsification transmits only coordinates with large magnitudes, low-rank methods approximate gradients with smaller matrices, and error feedback accumulates residuals not transmitted in the current iteration for use in later iterations (Konečný et al., 2016; Bernstein et al., 2018; Alistarh et al., 2017; Lin et al., 2018; Stich et al., 2018; Karimireddy et al., 2019; Vogels et al., 2019). However, magnitude sparsification mainly retains components with large numerical values and may overlook components that have

small magnitudes, but are important to the current task. Although quantization can reduce the number of bits used to represent each value, it generally does not change the dimensionality of the state to be transmitted and may introduce quantization errors. In view of these limitations, this paper focuses on compression methods that reduce the transmission dimension through low-dimensional representations.

2.2 Existing Compression Methods

Among the classical dimensionality-reduction methods most relevant to our baselines are PCA and linear autoencoders (Jolliffe & Cadima, 2016; Baldi & Hornik, 1989). PCA determines a fixed low-dimensional subspace from data variance, while a linear autoencoder learns a linear compression mapping through a reconstruction objective. Both are nevertheless limited by linear expressiveness and have difficulty describing complex nonlinear structures in training states. Research on invertible neural networks and normalizing flows (Rezende & Mohamed, 2015; Dinh et al., 2017; Kingma & Dhariwal, 2018) has shown that structured nonlinear transformations can reorganize the local geometry of data. CINN draws on this idea by applying a dimension-preserving nonlinear transformation before forced dimensionality reduction. This makes the local subspace observed by the low-dimensional projection vary with the input, allowing more optimization directions useful for model updates to be transmitted under a limited communication budget.

3 Method

3.1 The Distributed-Training Communication Bottleneck and Limitations of Existing Compression Methods

Distributed pre-training repeatedly transfers model weights and parameter fragments between devices. This paper uniformly denotes the data that cross these communication boundaries as the training state

$$\boldsymbol{x}_t \in \mathbb{R}^N. \tag{3.1}$$

Here, t is the training-iteration index, $\boldsymbol{x}_t$ is the state transferred at iteration t, and N is its dimension.

If one batch contains B states and each coordinate occupies q bytes, direct transmission requires

$$M_{\mathrm{raw}} = BNq. \tag{3.2}$$

Here, M_{raw} is the uncompressed data volume.

The time required for one communication operation can be approximated as

$$T_{\mathrm{comm}} \approx \alpha + \frac{8M_{\mathrm{raw}}}{\beta}. \tag{3.3}$$

Here, α is the fixed startup and synchronization latency, β is the effective link rate, and the factor 8 converts bytes to bits. As the state dimension, batch size, and number of devices increase, communication may enter the critical training path and leave devices idle.

Current mainstream compression methods, such as PCA and linear compression, use linear subspaces whose representational capacity is limited by a fixed linear structure. A fixed linear compressor cannot continuously preserve task-sensitive directions that vary across samples and training stages. The following theorem describes this limitation.

Theorem 1 (Loss of Task-Sensitive Directions under Linear Compression) *Let the linear compression be $\boldsymbol{z} = W\boldsymbol{x}$, where $W \in \mathbb{R}^{K\times N}$, $K < N$, and $\mathrm{rank}(W) = K$. If the gradient of the task loss at a state $\boldsymbol{x}$ contains a nonzero component in the null space of W, then this component can change the task loss under a first-order approximation without changing the compressed representation $\boldsymbol{z}$.*

In Theorem 1, $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{z} \in \mathbb{R}^K$ are the states before and after compression, respectively, and W is the rank-K compression matrix.

Proof: By the rank–nullity theorem,

$$\dim \ker(W) = N - \mathrm{rank}(W) = N - K > 0. \tag{3.4}$$

Here, $\ker(W) = \{\boldsymbol{v} \in \mathbb{R}^N \mid W\boldsymbol{v} = \mathbf{0}\}$ is the null space of W. Since $K < N$, the compressor necessarily has nonzero invisible directions.

Let $\boldsymbol{g}_x = \nabla_{\boldsymbol{x}} \mathcal{L}_{\text{task}}$, and denote its orthogonal projection onto $\ker(W)$ by

$$\boldsymbol{v} = P_{\ker} \boldsymbol{g}_x \neq \mathbf{0}, \qquad P_{\ker} = I_N - W^{\mathsf{T}}(WW^{\mathsf{T}})^{-1}W. \tag{3.5}$$

Here, $\boldsymbol{g}_x$ is the task gradient, $P_{\ker}$ is the orthogonal projector onto $\ker(W)$, and I_N is the identity matrix. Because W has full row rank, WW^{T} is invertible.

Because $\boldsymbol{v} \in \ker(W)$, $W\boldsymbol{v} = \mathbf{0}$. Hence, for a sufficiently small perturbation $\eta\boldsymbol{v}$,

$$\begin{aligned} W(\boldsymbol{x} + \eta\boldsymbol{v}) &= W\boldsymbol{x}, \\ \mathcal{L}_{\text{task}}(\boldsymbol{x} + \eta\boldsymbol{v}) - \mathcal{L}_{\text{task}}(\boldsymbol{x}) &= \eta \left\| P_{\ker} \boldsymbol{g}_x \right\|_2^2 + o(\eta). \end{aligned} \tag{3.6}$$

Here, $\eta > 0$ is the perturbation magnitude and $o(\eta)$ is a higher-order remainder. Thus, the compressed representation is unchanged while the task loss changes to first order, proving Theorem 1.

When task-sensitive directions vary with samples or training stages, a single K-dimensional subspace cannot preserve them consistently. CINN therefore introduces an input-dependent nonlinear transformation before low-dimensional projection.

3.2 Overall Design of CINN

Given a communication budget of K coordinates, CINN maps an N-dimensional training state to a compact representation while preserving task-relevant variations. It first applies a dimension-preserving nonlinear transformation $f_\theta : \mathbb{R}^N \to \mathbb{R}^N$ and then performs linear projection:

$$\boldsymbol{u} = f_\theta(\boldsymbol{x}), \qquad \boldsymbol{c} = W\boldsymbol{u} = W f_\theta(\boldsymbol{x}). \tag{3.7}$$

Here, $\boldsymbol{u} \in \mathbb{R}^N$ is the transformed state, $W \in \mathbb{R}^{K \times N}$ is a learnable projection matrix, and $\boldsymbol{c} \in \mathbb{R}^K$ is the transmitted representation, with $K < N$.

The theoretical data volume after compression and the coordinate compression factor are

$$M_{\text{cmp}} = BKq, \qquad \rho = \frac{M_{\text{raw}}}{M_{\text{cmp}}} = \frac{N}{K}. \tag{3.8}$$

Here, M_{cmp} is the compressed communication volume and ρ is the theoretical coordinate compression factor.

As shown in Figure 1, CINN first reorganizes the local geometry of the training state through a dimension-preserving nonlinear transformation and then performs low-dimensional projection.

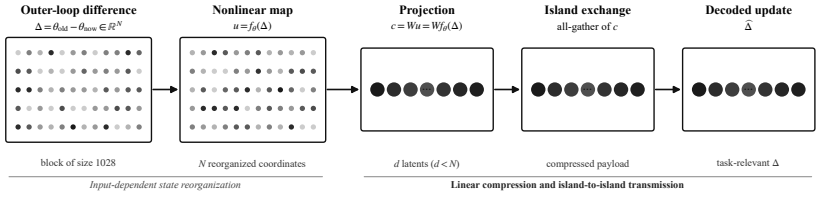


Figure 1: CINN pipeline.

3.3 The Dimension-Preserving Nonlinear Transformation and Its Local Null Space

To make the directions observed by the projection depend on the input state, CINN applies a dimension-preserving nonlinear transformation before projection. The following theorem establishes its mathematical feasibility.

Theorem 2 (Input-Dependent Local Null Space of CINN) *Let $C(\boldsymbol{x}) = W f_\theta(\boldsymbol{x})$, where f_θ is differentiable in a neighborhood of $\boldsymbol{x}$ and its Jacobian matrix $J_f(\boldsymbol{x})$ is invertible. Then the local null space of CINN at $\boldsymbol{x}$ satisfies*

$$\ker(J_C(\boldsymbol{x})) = J_f(\boldsymbol{x})^{-1}(\ker(W)), \qquad \dim\ker(J_C(\boldsymbol{x})) = N - K. \tag{3.9}$$

Here, $C : \mathbb{R}^N \to \mathbb{R}^K$ is the complete encoder, while $J_C(\boldsymbol{x})$ and $J_f(\boldsymbol{x})$ are the Jacobians of C and f_θ, respectively. The notation $J_f(\boldsymbol{x})^{-1}(\ker(W))$ denotes the preimage of $\ker(W)$ under $J_f(\boldsymbol{x})$.

Proof: By the chain rule, $J_C(\boldsymbol{x}) = W J_f(\boldsymbol{x})$. For any direction $\boldsymbol{v} \in \mathbb{R}^N$,

$$\begin{aligned} \boldsymbol{v} \in \ker(J_C(\boldsymbol{x})) &\iff W J_f(\boldsymbol{x})\boldsymbol{v} = \boldsymbol{0} \\ &\iff J_f(\boldsymbol{x})\boldsymbol{v} \in \ker(W) \\ &\iff \boldsymbol{v} \in J_f(\boldsymbol{x})^{-1}(\ker(W)). \end{aligned} \tag{3.10}$$

Because an invertible linear transformation preserves subspace dimension, $\dim\ker(J_C(\boldsymbol{x})) = N - K$, proving Theorem 2.

Theorem 2 does not remove the null space required by dimensionality reduction. Instead, the input-dependent $J_f(\boldsymbol{x})$ changes its local orientation with $\boldsymbol{x}$, allowing currently important directions to move out of it.

The dimension-preserving transformation is implemented with residual blocks:

$$\boldsymbol{h}_{l+1} = \boldsymbol{h}_l + \boldsymbol{\gamma}_l \odot F_l(\mathrm{LN}(\boldsymbol{h}_l)), \qquad \boldsymbol{h}_0 = \boldsymbol{x}. \tag{3.11}$$

Here, l indexes the blocks; $\boldsymbol{h}_l \in \mathbb{R}^N$ is the block state; F_l is a nonlinear subnetwork; LN denotes layer normalization; $\boldsymbol{\gamma}_l \in \mathbb{R}^N$ is a learnable residual-scaling vector; and $\odot$ denotes elementwise multiplication. Their composition forms f_θ.

3.4 Jacobian Reward Function and Preservation of Task-Sensitive Directions

The dimension-preserving nonlinear map structure enables adaptive preservation but does not identify which directions matter to the task. The task gradient $\boldsymbol{g}_x$ is therefore used to guide the transformation. To compare directions independently of gradient magnitude, define

$$\widehat{\boldsymbol{g}}_x = \frac{\boldsymbol{g}_x}{\|\boldsymbol{g}_x\|_2 + \varepsilon}. \tag{3.12}$$

Here, $\widehat{\boldsymbol{g}}_x$ is the normalized task direction and $\varepsilon > 0$ prevents division by zero.

Then the following result holds.

Theorem 3 (Jacobian Response and Visibility of the Task Direction) *Let the encoder C be differentiable at the state $\boldsymbol{x}$, and let $\widehat{\boldsymbol{g}}_x$ be a normalized task direction. For a sufficiently small real number η,*

$$\|C(\boldsymbol{x} + \eta\widehat{\boldsymbol{g}}_x) - C(\boldsymbol{x})\|_2^2 = \eta^2 \|J_C(\boldsymbol{x})\widehat{\boldsymbol{g}}_x\|_2^2 + o(\eta^2). \tag{3.13}$$

Here, η is a small perturbation magnitude and $o(\eta^2)$ is negligible relative to η^2 as η approaches zero.

Proof: By the first-order Taylor expansion,

$$C(\boldsymbol{x} + \eta\widehat{\boldsymbol{g}}_x) = C(\boldsymbol{x}) + \eta J_C(\boldsymbol{x})\widehat{\boldsymbol{g}}_x + o(\eta). \tag{3.14}$$

Moving $C(\boldsymbol{x})$ to the left-hand side gives

$$C(\boldsymbol{x} + \eta\widehat{\boldsymbol{g}}_x) - C(\boldsymbol{x}) = \eta J_C(\boldsymbol{x})\widehat{\boldsymbol{g}}_x + o(\eta). \tag{3.15}$$

Computing the squared 2-norm yields

$$\|C(\boldsymbol{x} + \eta\widehat{\boldsymbol{g}}_x) - C(\boldsymbol{x})\|_2^2 = \eta^2 \|J_C(\boldsymbol{x})\widehat{\boldsymbol{g}}_x\|_2^2 + o(\eta^2). \tag{3.16}$$

This is Equation (3.13), proving Theorem 3.

Theorem 3 shows that $\|J_C(\boldsymbol{x})\widehat{\boldsymbol{g}}_x\|_2^2$ measures the local sensitivity of the transmitted code to a task-sensitive direction. The Jacobian reward is therefore defined as

$$\mathcal{Q}_{\text{jac}} = \mathbb{E}_{\boldsymbol{x}}\left[\|J_C(\boldsymbol{x})\,\text{stopgrad}(\widehat{\boldsymbol{g}}_x)\|_2^2\right]. \tag{3.17}$$

Here, $\mathbb{E}_{\boldsymbol{x}}$ denotes expectation over training states and $\text{stopgrad}(\cdot)$ prevents differentiation through task-gradient estimation. A larger reward increases the encoder response to task-relevant directions.

3.5 Low-Dimensional Compression Based on Orthogonal Projection

After nonlinear reorganization, W projects the state from N to K dimensions. To prevent multiple transmitted coordinates from repeatedly representing the same direction, the following constraint is imposed:

$$\mathcal{R}_{\text{orth}} = \left\|WW^{\mathsf{T}} - I_K\right\|_F^2. \tag{3.18}$$

Here, $\|\cdot\|_F$ is the Frobenius norm. As WW^{T} approaches I_K, the rows of W become mutually orthogonal, reducing redundancy among transmitted coordinates.

The orthogonality loss therefore improves utilization of the limited K-dimensional transmitted space.

3.6 Overall CINN Architecture and Joint Loss Function

CINN combines three components: the task loss supports the downstream objective, the Jacobian reward strengthens the encoder response to task-sensitive directions, and the orthogonality regularizer reduces redundancy among transmitted coordinates. The joint objective is

$$\mathcal{L}_{\text{CINN}} = \mathcal{L}_{\text{task}} - \lambda_{\text{jac}}\mathcal{Q}_{\text{jac}} + \lambda_{\text{orth}}\mathcal{R}_{\text{orth}}. \tag{3.19}$$

Here, $\mathcal{L}_{\text{task}}$ is the loss of downstream tasks, $\mathcal{Q}_{\text{jac}}$ is the Jacobian reward, $\mathcal{R}_{\text{orth}}$ is the orthogonality term, and $\lambda_{\text{jac}}, \lambda_{\text{orth}} \geq 0$ are their weights. The Jacobian term is negative because the total objective is minimized while the reward is maximized.

4 Experiments

4.1 Experimental Setup

The experiments use two NVIDIA GPUs as two DiLoCo islands, ResNet-56 as the shared backbone, and CIFAR-10 as the dataset. Each GPU holds a replica of the same model and is trained with SGD (Nesterov momentum 0.9, weight decay 5×10^{-4}), inner learning rate 0.05, cosine annealing over the training epochs, and batch size 128. After $H{=}10$ local inner steps with no cross-island communication, the islands exchange the outer-loop parameter difference accumulated over those steps—the intermediate payload that DiLoCo actually sends—and apply it with outer learning rate 0.5. This difference is split into blocks of size 1028. PCA fits a frozen linear subspace on warmup blocks and then projects each block to d coordinates. CINN instead applies an input-dependent nonlinear map followed by a learnable projection to d, with Jacobian–gradient response weight $\lambda_{\text{jac}}{=}0.1$ and orthogonality weight $\lambda_{\text{orth}}{=}10^{-2}$; a decoder reconstructs the block after the island exchange. Linear projection uses a trainable $1028 \to d \to 1028$ map. CINN, PCA, linear projection, and uncompressed direct transfer of this parameter difference are compared.

4.2 CIFAR-10 Compression Results

Table 1 and Figure 2 compare CINN at $d{=}128$ with uncompressed direct transmission over 200 epochs under the same inner and outer learning rates.

Table 1: CIFAR-10 DiLoCo test accuracy versus epoch: direct transmission and CINN at d=128

Method	20	40	60	80	100
Direct transmission	80.26%	85.72%	86.71%	88.62%	86.93%
CINN (d=128)	68.45%	77.54%	81.41%	82.82%	82.96%
Method	**120**	**140**	**160**	**180**	**200**
Direct transmission	88.92%	89.73%	90.14%	90.07%	**90.16%**
CINN (d=128)	81.37%	83.23%	84.66%	85.21%	85.44%

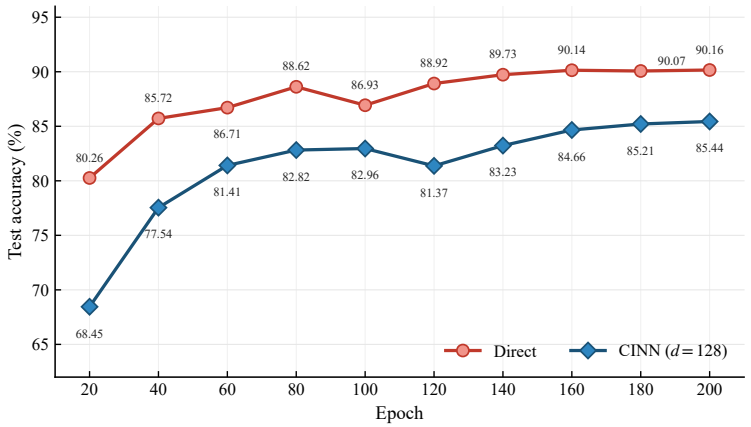


Figure 2: CIFAR-10 DiLoCo test accuracy versus epoch for direct transmission and CINN at d=128.

Direct transmission starts 11.81 points higher (80.26% versus 68.45%) because it is uncompressed. CINN at d=128 rises faster under an $8\times$ smaller outer-loop payload, from 68.45% to 85.44%, and by epoch 200 the gap has narrowed to 4.72 points (90.16% versus 85.44%). The compressed outer-loop difference still supports continued learning, and CINN closes on uncompressed transmission by epoch 200.

The same outer-loop difference is then compressed to $d \in \{16, 32, 64, 128\}$ and trained for 200 epochs, comparing CINN with PCA and linear projection at the same bottleneck. Table 2 reports the resulting test accuracy.

Table 2: CIFAR-10 DiLoCo test accuracy (200 epochs, sync every 10 steps)

Method	d=16	d=32	d=64	d=128
PCA	55.32%	68.41%	78.21%	83.74%
Linear projection	62.80%	70.75%	79.68%	84.50%
CINN	**72.78%**	**77.69%**	**82.57%**	**85.44%**

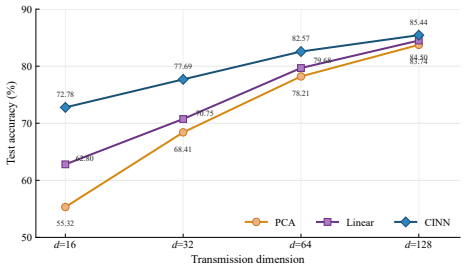


Figure 3: CIFAR-10 test accuracy versus transmission dimension.

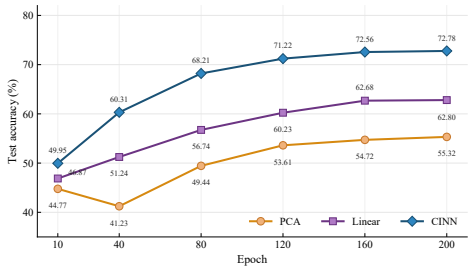


Figure 4: CIFAR-10 test accuracy at d=16 versus epoch.

The test accuracies corresponding to Table 2 are shown in Figure 3. CINN attains the highest accuracy at every transmission dimension, and the advantage is most pronounced under the strongest compression. At d=16, CINN reaches 72.78%, 9.98 points above linear projection and 17.46 points above PCA. This gap shows that CINN still extracts a usable representation of the outer-loop difference when the bottleneck is most severe. As the latent dimension increases, CINN remains the best method throughout, reaching 85.44% at d=128. The result indicates that the input-dependent nonlinear encoder learns the compressed update more effectively than fixed PCA or a supervised linear map, especially at high compression ratios.

Figure 4 presents a line chart of the d=16 accuracies over epochs 10, 40, 80, 120, 160, and 200. CINN rises fastest under this ultra-high compression. By epoch 40 it already reaches 60.31%, and by epoch 80 it attains 68.21%, while linear projection and PCA remain at 56.74% and 49.44%. The subsequent gains stay with CINN: it continues to climb to 72.78% at epoch 200, 9.98 points above linear projection and 17.46 points above PCA. These curves show that CINN learns the compressed outer-loop update more rapidly, and that this advantage persists throughout training.

4.3 Comparison of Theoretical Payload and Transmission Speed

This subsection reports the theoretical float32 payload of one DiLoCo outer-loop update for CIFAR-10 ResNet-56, together with the measured two-GPU `all_gather` latency of that payload. Uncompressed direct transmission sends all 855,770 coordinates (3,423,080 bytes). CINN splits the update into blocks of size 1028 and transmits a d-dimensional latent plus one scale per block. An `all_gather` of each payload is timed between two GPUs, and the median latency over 100 repeats is reported. To mimic a long-haul island-to-island link, we inject a communication delay $T = \mathrm{RTT} + 8M/\beta$ after each outer-loop synchronization, with $\mathrm{RTT} = 200\,\mathrm{ms}$ and $\beta = 1\,\mathrm{Gbps}$. Table 3 lists both quantities at the dimensions used in the experiments.

Table 3: Payload and two-GPU `all_gather` latency of one ResNet-56 outer-loop update

Transmission dimension	Payload	Compression	Reduction	Latency (ms)	Speedup
Uncompressed (855,770)	3,423,080 B	1×	0%	2.495	1.00×
d=128	429,828 B	8.0×	87.45%	0.298	8.37×
d=64	216,580 B	15.8×	93.67%	0.194	12.86×
d=32	109,956 B	31.1×	96.79%	0.100	24.95×
d=16	56,644 B	60.4×	98.35%	0.066	37.80×

As Table 3 shows, compressing the outer-loop update to d=128 reduces the payload by 8.0× and the two-GPU `all_gather` latency from 2.495 ms to 0.298 ms (8.37×). Further compression continues to shorten the transfer, reaching 24.95× at d=32 and 37.80× at d=16. Below a few tens of kilobytes the speedup lags the coordinate ratio, because startup and synchronization begin to dominate.

To measure larger payloads, the two GPUs are run through an actual DiLoCo training path. Each outer-loop synchronization is delayed as if a 1-billion-parameter FP16 update were sent over a 200 ms, 1 Gbps link. Direct transmission therefore carries 2000 MB per sync; CINN reduces that payload to $1/8$, $1/16$, and $1/32$. The first two epochs are used for warm-up, after which the wall-clock time is measured over ten additional epochs. Table 4 reports the mean epoch time over these ten measured epochs.

Table 4: Mean epoch wall-clock time under a simulated 1B-FP16 WAN payload

Method	Payload	Mean epoch time (s)	Speedup vs. Direct
Direct transmission	2000 MB	346.1	1.00×
CINN (1/8)	250 MB	62.6	5.5×
CINN (1/16)	125 MB	42.2	8.2×
CINN (1/32)	62.5 MB	31.9	10.9×

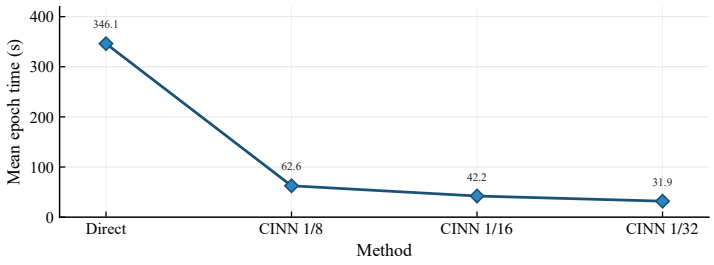


Figure 5: Mean per-epoch wall-clock time of a 1B-parameter FP16 payload under a 200 ms / 1 Gbps WAN model. Direct transmission takes 346.1 s per epoch; CINN at compression ratios $1/8$, $1/16$, and $1/32$ reduces this to 62.6 s, 42.2 s, and 31.9 s, respectively.

As Table 4 and Figure 5 show, the reported runtime is averaged over ten measured epochs after two warm-up epochs. CINN reduces the mean epoch time from 346.1 s for direct transmission to 62.6 s, 42.2 s, and 31.9 s at compression ratios of $1/8$, $1/16$, and $1/32$, respectively. Even without overlapping the previous encoding step, CINN shortens the epoch as the uncompressed payload grows and the link becomes the bottleneck.

5 Conclusion

This paper studies the remaining communication volume after DiLoCo has already reduced the *frequency* of island-to-island synchronization. Linear compression of the transferred training state necessarily leaves a null space: a task-sensitive direction in that subspace can change the loss to first order without changing the transmitted code. CINN addresses this limitation with an input-dependent nonlinear transformation before a compressive projection, so that the local invisible subspace can be reoriented with the current state. Jacobian–gradient response strengthens the first-order response of the code to loss-sensitive directions, and an orthogonality regularizer reduces redundancy among the few transmitted coordinates.

The CIFAR-10 DiLoCo experiments support this design. Under an $8\times$ smaller outer-loop payload, CINN still learns a usable task update and remains above PCA and linear projection at the same bottleneck, with the largest margin at the strongest compression. Because the exchanged tensor is much smaller, the same outer step also finishes faster on a high-latency, bandwidth-limited link. CINN therefore converts a limited communication budget into task-relevant coordinates, cutting both the per-transfer payload and the waiting time of cluster synchronization without discarding the information needed for training.

Use of AI

In this work, generative AI tools were used to assist with research ideation and execution, including feedback on experimental presentation and the implementation of plotting and experiment-support scripts, and to assist with translation of selected passages. Generative AI tools were also used to aid and polish writing, to draft or revise selected passages, to retrieve and check related work and citation coverage, to edit figures and LaTeX formatting, and to improve readability. Generative AI tools were not used to generate synthetic datasets or to formulate or prove mathematical claims. All AI-assisted text, code, and figures were reviewed by the authors; experimental numbers and theoretical claims were checked against the original logs and proofs. The authors take responsibility for the final content of this work.

References

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, 2017.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 1989.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 2012.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Johan Donchez, A. Rannen-Triki, A. Shaker, Razvan Pascanu, et al. DiLoCo: Distributed low-communication training of language models. *arXiv:2311.08105*, 2023.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017.

Ian T. Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A*, 2016.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 2018.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv:1610.05492*, 2016.

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Joskovec, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

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

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv:1802.05799*, 2018.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems*, 2018.

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