The Journal of Supercomputing (2025) 81:964
https://doi.org/10.1007/s11227-025-07451-z

Balancing communication overhead and accuracy in compression integration: a survey

Aiqiang Yang[1] · Jie Liu[1,2,3] · Bo Yang[1,2] · Zeyao Mo[4] · Keqin Li[5]

Accepted: 12 May 2025

Abstract
In large-scale distributed training, communication compression techniques are widely used to reduce the significant communication overhead caused by the frequent exchange of model parameters or gradients between training nodes. However, these techniques often introduce additional computational complexity and may lead to data loss, thereby affecting model convergence and performance. This review examines key optimization methods in communication compression, including pruning techniques that remove irrelevant weights, quantization techniques that convert floating-point parameters to low-precision representations, and sparsification techniques that transmit only critical gradients. Low-rank approximation techniques, which compress parameters through matrix factorization, are particularly useful for large-scale models. These techniques have also been widely applied in various application scenarios, demonstrating their effectiveness in different environments. Application scenarios include distributed training, federated learning, and edge computing, where bottlenecks are carefully identified and evaluated in common scenarios, providing a basis for further optimization. Future development directions emphasize co-design of hardware and algorithms, dynamic strategies, and cross-layer optimization. This study provides valuable comparisons of key methods and theoretical analysis for efficient communication compression in distributed systems.

Keywords Communication compression · Collective communication · Distributed training · Stochastic gradient

1 Introduction

Large-scale distributed training has become the cornerstone of modern machine learning, enabling the training of increasingly complex models through the use of multiple parallel computing nodes. Unlike traditional distributed learning, which

Jie Liu, Bo Yang, Zeyao Mo and Keqin Li have contributed equally to this work.

Extended author information available on the last page of the article

Published online: 04 June 2025

primarily relied on parameter servers or synchronous updates with limited scalability [1], modern approaches emphasize decentralized architectures and efficient communication to handle massive datasets and deep neural networks. For example, early implementations of distributed stochastic gradient descent (SGD) [2] required synchronizing full gradients across nodes at each iteration, whereas modern frameworks adopt advanced decentralized strategies [3] and gradient compression techniques [4] to minimize latency and bandwidth consumption.

However, as model sizes have exponentially grown from ResNet's 25 million parameters to approximately 1.8 trillion parameters in today's GPT-4 [5], the volume of data exchanged during training (such as gradients and parameters) has created significant redundancy, slowing down convergence and increasing costs [6]. To address this issue, communication compression techniques have emerged. In their seminal work *Parallel and Distributed Computation* [7], Bertsekas et al. proposed scalar quantization of gradients in distributed optimization, mapping continuous values to discrete intervals, which laid an important foundation for communication compression theory. Subsequent research, such as 1-bit SGD [4], further advanced this idea but also revealed two persistent bottlenecks: first, the trade-off between communication overhead and computational cost—compression reduces the amount of transmitted data but increases local computational burden; second, the trade-off between compression efficiency and model accuracy—many lossy methods may lose critical information.

To tackle these challenges, researchers have developed a variety of methods. Quantization sacrifices precision for bandwidth savings; sparsification, which transmits only important or nonzero data elements while ignoring all zero or insignificant values [8], reduces the volume of data transmitted; low-rank decomposition [9] reduces the dimensionality of weight matrices; pruning [10] removes redundant weights; and hybrid techniques such as quantized sparsification [11] further optimize overall performance.

These techniques have been widely applied in scenarios such as distributed training, federated learning, collective communication, and IoT. In distributed training, the sparse Allreduce method [12] optimizes communication through Top-K value selection; gradient compression [13] accelerates training and reduces communication needs, but must be flexibly adjusted according to task requirements and environmental conditions; and for non-convex smoothing problems, relevant algorithms [14] have achieved near-optimal convergence rates and established theoretical lower bounds. In federated learning, unstable networks [15] and noisy channels [16] degrade performance, but decentralized training [17] and adaptive methods [18] can reduce communication while maintaining performance and convergence. Meanwhile, traditional communication libraries such as NCCL optimize underlying transmission but lack built-in compression capabilities, prompting frameworks like BytePS [19] to integrate compression mechanisms on their basis.

Looking ahead, from high-density AI training with trillion-parameter models to edge-cloud collaboration and emerging paradigms like quantum distributed learning, adaptive compression techniques will be essential for handling heterogeneous hardware and massive data streams. This makes it crucial to systematically evaluate the strengths and weaknesses of existing methods for future development.

The remainder of this paper is structured as follows. Section 2 outlines the two major bottlenecks in communication compression. Section 3 delves into the primary methods of communication compression. Section 4 offers specific solutions for these bottlenecks. Section 5 describes the typical application scenarios of communication compression. Section 6 summarizes the relevant communication methods. Finally, Sect. 7 concludes the paper. Figure 1 shows a brief introduction of the whole process.

2 Bottleneck

This section delves into the two major bottlenecks of communication compression in distributed deep neural network training: the trade-off between communication overhead and computational costs, and the balance between compression efficiency and model accuracy. Communication overhead is a primary bottleneck in distributed training, especially in large-scale settings where frequent exchanges of model parameters significantly increase communication costs. Many solutions attempt to reduce these costs through data compression, but this often introduces additional computational burdens and may affect model convergence and accuracy. Moreover, achieving fast model deployment in resource-constrained environments, such as mobile devices and embedded systems, hinges on striking a balance between compression efficiency and maintaining model accuracy.

2.1 Communication overhead and computational costs

Communication overhead is a key bottleneck in distributed deep neural network (DNN) training. High communication costs arise when model parameters need to be exchanged frequently across the network, especially in large-scale settings with many working nodes. At the moment, a lot of solutions concentrate on cutting down

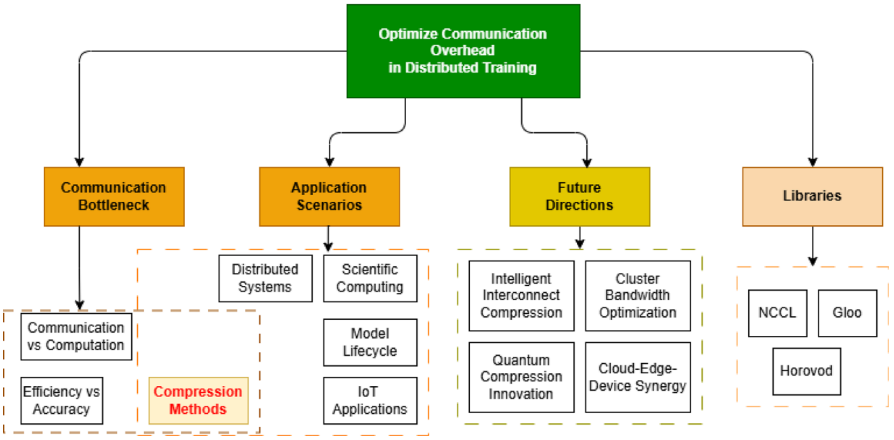


Fig. 1 Overview of communication compression

on communication overhead without realizing that doing so also increases computing overhead. This is particularly true when training complex models on large datasets like ImageNet, for instance, ResNet-50. While strategies like the Stich method reduce communication costs by locally aggregating updates before sharing, they may introduce additional computational overhead and can significantly reduce model accuracy [20]. This research investigates the use of communication compression methods like quantization [21] and sparsification in decentralized parallel stochastic gradient descent (D-PSGD). It shows that while compression reduces communication costs, it can lead to increased computational complexity, especially when trying to maintain the correctness and convergence of the algorithms [22].

Without sacrificing any information, a lossless homomorphic compression algorithm lowers the amount of communication data. This method involves two phases: compression and recovery, aiming to balance the benefits of compression with the computational overhead. However, it highlights the increased complexity in computational processes involved in these two steps [23]. Adaptive compression mechanisms like Lazy Aggregation (LAG) and CLAG aim to optimize communication by selectively compressing data. The authors note that while these methods can reduce communication frequency, they add computational overhead due to the need for continuous adjustments and checks to determine when to compress or skip communication [24].

A theoretical examination of the trade-offs between compute and communication in communication compression-based distributed learning algorithms was presented by Huang et al. [14]. They establish that while compression can reduce communication overhead, it often requires more computational steps to achieve similar convergence rates, thereby increasing overall computational costs.

The current Gradient Compression (GC) algorithm applied in a hierarchical approach reduces the duration of communication, but introduces a significant compression overhead [25], which consequently decreases overall training efficiency. In their discussion of a hybrid communication compression technique, Chen et al. [26] emphasized the trade-off between lowering communication overhead and raising the computational complexity of handling compressed data during training; this study [27] introduced adaptively compressed stochastic gradient descent (AC-SGD) to balance communication efficiency and computational load. It emphasizes that adaptive compression methods, while effective at reducing communication frequency, require additional computational resources to manage the dynamic compression levels.

It has been demonstrated that techniques like stochastic gradient compression [28] greatly lower communication overheads [29], but they can also lead to considerable accuracy degradation and increased computational costs. Since both gradient computation and gradient compression are computationally demanding operations with high compression overheads [30] that may cause a slowdown overall, gradient compression techniques [31] are not appropriate for overlapping with gradient computation.

Additionally, gradient compression [32] significantly lowers the communication cost, but comes with a non-negligible overhead that prevents scalability and slows down training in real-world applications. A unifying framework known as GRACE was created by Xu et al. [33] and supports a number of gradient compression

techniques. Nevertheless, in comparison with training ResNet-50 [34] without gradient compression, the findings demonstrate that only random k (sending random k elements of the gradient) achieves superior throughput. All other compression techniques, however, work noticeably worse. The performance is worse than it would be if compression weren't used since the overhead of compression outweighs the time savings in communication. The main features and performance metrics of these methods are summarized in Table 1.

2.2 Balance between compression efficiency and model accuracy

One of the main obstacles to ensuring fast model deployment which is especially important in resource-constrained situations like mobile devices and embedded systems is striking a balance between compression efficiency and model correctness.

Model accuracy is a key metric for measuring model performance on a given task, and the impact of compression distortion on accuracy permeates the entire training process. First, in terms of gradient fidelity, quantization or sparsification introduces noise, causing the parameter update direction to deviate from the true gradient. This misalignment directly disrupts the consistency of the optimization trajectory. Second, regarding convergence quality, compression operations (such as high proportions of gradient discarding) can significantly slow down the convergence speed. Especially in the early stages of training, more iterations are required to reach the target accuracy. Moreover, improper adjustment of dynamic compression strategies may trap the model in local optima. Finally, in terms of task accuracy, excessive compression can damage the model's final performance. These phenomena reveal that compression techniques, by interfering with gradient statistical properties, simultaneously affect the three dimensions of accuracy metrics—task accuracy, convergence quality, and gradient fidelity. Therefore, this multidimensional perspective demands explicit metric selection and scenario adaptation to achieve Pareto optimality between communication efficiency and model performance.

At the same time, the synergy between data exchange frequency and compression strategies is crucial for balancing accuracy, scalability, and efficiency. High-frequency synchronization reduces communication volume by 90% through lightweight methods while maintaining convergence quality via error feedback. Low-frequency scenarios adopt extreme quantization and residual encoding to cut energy consumption. Adaptive strategies dynamically adjust compression ratios and synchronization intervals based on gradient variance or network bandwidth, achieving corresponding efficiency improvements in heterogeneous environments. These strategies show that the co-design of compression and frequency control, whether prioritizing speed for GPU clusters or scalability for edge devices, plays an important role in communication compression in distributed training scenarios.

Communication delay was integrated with probabilistic quantization and random sparsification by Konevcny et al. [35], and their method also mixes communication delay with (random) quantization and sparsification. While the method produces considerable compression benefits on a certain CNN and LSTM model, it also causes a notable slowdown in convergence speed, which in turn lowers accuracy; the 3LC

Table 1 Communication costs vs. computation

Technique/Method	Main Focus	Pros	Cons
Stich Method	Reduces communication costs by local aggregation	Significant reduction in communication costs	Increases computational overhead, accuracy loss
Quantization & Sparsification (D-PSGD)	Reduces communication cost but increases computational complexity	Lower communication costs, high scalability	Increases computational complexity, slows convergence
Lossless Homomorphic Compression	Balances compression benefits with computational overhead	Lossless compression without sacrificing information	Increased complexity in compression and recovery phases
Lazy Aggregation (LAG) & CLAG	Optimizes communication by selective compression but adds overhead	Reduced communication frequency, adaptable	Adds continuous adjustments, computational overhead
Gradient Compression (GC)	Reduces communication duration but introduces compression overhead	Reduced communication duration	Increased compression overhead reduces efficiency
AC-SGD	Balances communication efficiency and computational load	Dynamic compression levels, efficient	Requires extra computational resources for dynamic adjustments
Stochastic Gradient Compression	Reduces communication overhead but leads to accuracy degradation	Greatly lowers communication costs	Accuracy degradation, higher computational cost
GRACE Framework	Framework supporting multiple gradient compression techniques	Unified framework, supports multiple techniques	Compression overhead outweighs communication savings

compression scheme balances traffic reduction, accuracy, and computation overhead [36]. The 3LC explores methods to mitigate these errors but acknowledges that aggressive compression can still lead to reduced accuracy; the study displays the deep gradient compression (DGC) technique, which significantly reduces communication bandwidth while implementing various strategies to preserve accuracy. However, it [37] demonstrated that if the compression parameters are not carefully tuned, the aggressive reduction in gradient information can lead to performance degradation.

In federated learning, the two-layer cumulative quantized compression approach improves transmission efficiency. The method reduced communication costs but faces challenges in maintaining model accuracy [38], emphasizing the trade-offs inherent in compression techniques. When using ScaleCom, it is crucial to be aware of the potential loss of model accuracy due to gradient compression [39]. When using large datasets and complex models in high-volume training scenarios, compression techniques can negatively impact model convergence and final accuracy. For example, employing a gradient compression strategy in a high-volume, multi-worker node training configuration for the Transformer model on the WMT14 En-De dataset [40] may cause significant accuracy loss. This occurs because gradient compression, while reducing communication data, may also remove important information needed for model training.

And the method put forward by strom [41] leads to significant accuracy degradation; TernGrad introduces accuracy loss (2% in GoogLeNet), computational overhead from ternary operations, theoretical reliance on gradient-bound assumptions, and implementation complexity from layer-wise scaling. [42]. Table 2 provides a comprehensive summary of these methods.

The significant drop in accuracy due to compression is an unresolved and incomplete bottleneck encountered, which can ultimately lead to poor accuracy of the results and make the entire experiment impossible to complete.

3 Optimization methods

In the realm of distributed deep learning, optimizing communication compression is crucial for enhancing efficiency and scalability. This chapter delves into a variety of optimization methods, including network pruning, weight sharing, quantization, sparsification, low-rank approximation, compression coding, and knowledge distillation. Each technique is designed to balance the trade-offs between reducing communication overhead and preserving model accuracy.

3.1 Communication compression types

Like many other machine learning models, deep neural networks have two stages: training and inference. In the training phase, the model's parameters (which, for neural networks, are primarily the network's weights) are discovered through data analysis. In the inference stage, fresh data are added to the model, and the outcomes are computed.

Table 2 Compression efficiency vs. accuracy

Technique/method	Main focus	Pros	Cons
Probabilistic quantization and sparsification	Balances communication delay, quantization, and sparsification	Compression benefits on CNN and LSTM models	Slows convergence speed, reduces accuracy
3LC compression scheme	Balances traffic reduction, accuracy, and computational overhead	Balances traffic reduction and computational overhead	Aggressive compression can still degrade accuracy
Deep gradient compression (DGC)	Reduces communication bandwidth while preserving accuracy	Significant bandwidth reduction, preserves accuracy	Requires careful tuning of compression parameters
Two-layer cumulative quantized compression	Improves transmission efficiency, but risks model accuracy	Reduces communication costs effectively	May cause loss of model accuracy
ScaleCom	Reduces communication cost but may reduce accuracy	Effective in reducing communication costs	Potential loss of accuracy with gradient compression
Gradient compression (Transformer Model-WMT14 En-De)	Reduces communication but risks performance degradation	Reduces communication data	May remove important information needed for training
Strom Method	Leads to accuracy degradation	Scalable for large datasets	Significant accuracy degradation
Ternary gradients, parameter localization, among others	Works for large-scale training but with potential loss	High communication efficiency, theoretical convergence, scalability	Up to 2% accuracy degradation in GoogLeNet

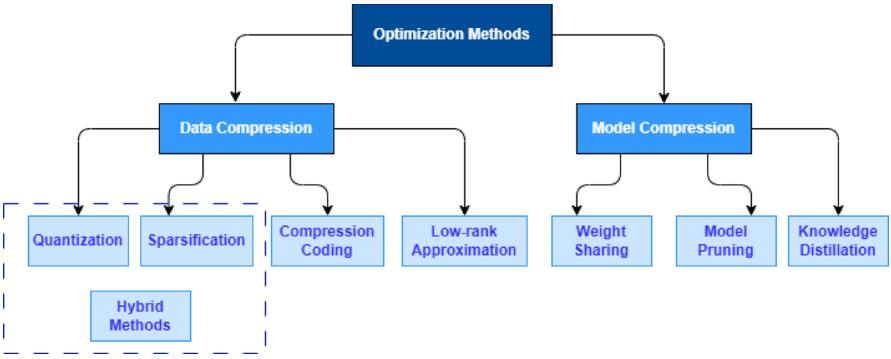


Fig. 2 Communication compression types

Over-parameterization, however, requires the use of many parameters during the training phase in order to capture minute details in the data. Once the training is complete and the model moves to the inference phase, we don't need as many parameters. The notion that we can streamline the model prior to deployment is reinforced by this supposition. The benefits of simplifying the model are numerous and include:

1. The reduction in calculation, which leads to shorter computation times and less power usage, is the most obvious advantage.
2. As the memory footprint shrinks, the model can operate on devices with less memory. The possibility of replacing the slow and energy-hungry DRAM with SRAM adds another performance advantage.
3. Updates and dissemination of applications benefit from smaller package sizes. For instance, several mobile phone markets limit the size of applications, and updates for cars can be made over-the-air (OTA) with smaller packages.

We can approach the enhancement methods from two perspectives: data compression and model compression. A detailed breakdown is provided in Fig. 2.

3.2 Network pruning

Networks often have more parameters than necessary, with many of them being redundant and removable. The main principle behind network pruning is to remove the least important components. Formally, this can be expressed as:

$$\min_{w} \left(L(D;w) + \lambda \|w\|_0 \right) \tag{1}$$

The symbols represent the following: min: The minimization operator, indicating that we are looking for the minimum value of the expression that follows; w: The vector of parameters or weights of the model that we are optimizing; $L(D; w)$: The loss function, which measures the discrepancy between the predicted values and the actual values in the data set D given the weights w; D: The set of data points or data set utilized to train the model; λ: A regularization parameter that modulates the

intensity of the penalty imposed on model complexity; $\|w\|_0$: The L_0 norm of the vector w, which counts the number of nonzero elements in w, represents a measure of sparsity in the weight vector.

It can be intuitively understood as balancing two goals:

1. Accuracy: Minimize prediction errors on the dataset D (measured by $L(D;w)$).
2. Simplicity: Reduce the number of nonzero weights (measured by the L_0 norm $\|w\|_0$, which counts nonzero values).

λ acts like a "knob" to adjust the trade-off: higher λ prioritizes simplicity over accuracy.

This equation encapsulates an optimization problem aimed at identifying the optimal set of weights w that minimizes the loss function over the data set D, concurrently integrating a regularization term to penalize model complexity, as measured by the L_0 norm of the weights. This type of problem is common in machine learning for promoting sparse solutions and preventing over fitting.

Alternatively, parameter pruning can be written in constrained optimization form:

$$\min_{w} L(D;w) + \lambda\|w\|_0 \quad \text{s.t.} \quad \|w\|_0 \leq \kappa \tag{2}$$

s.t.: An abbreviation for "subject to," indicating a constraint on the optimization problem; $\leq$: The less than or equal to symbol;

κ: A constant denoting the upper limit on the permissible number of nonzero entries within the weight vector w.

In deep learning model optimization, a pretrained model serves as the foundation. Optimization begins with evaluating the importance of the weights and neurons by analyzing the absolute magnitude of the weights and the activation output of the neurons [43]. Weights close to zero or low neuron output indicate relative unimportance, which can be assessed quantitatively evaluated using L1 or L2 norms [44].

Algorithm 1 implements the principles of Eq. 1 and 2 through practical approximations of the L_0-norm constraint:

Principle Remove weights with the smallest absolute values (assuming near-zero weights contribute minimally to predictions).

Algorithm 1 Magnitude-based weight pruning

Input: Model model, Pruning rate pruning_rate
Output: Pruned model
Collect all weights W from the model;
Compute the threshold τ as the pruning_rate-th quantile of $|W|$;
for *each parameter param in model* **do**
  Compute mask M where $M_{ij} = 1$ if $|\text{param}_{ij}| > \tau$ and 0 otherwise;
  Apply mask: param $\leftarrow$ param $\odot M$;
end
return model;

Model pruning entails the elimination of less significant weights or neurons, with the objective of diminishing model complexity and the count of parameters [45]. After pruning [46], the model may need to recover or improve performance through fine-tuning on specific datasets and adjusting the remaining parameters to fit the new network structure.

An incremental pruning approach is frequently employed, where a small number of parameters are gradually removed and fine-tuned to evaluate the performance impact, thereby avoiding performance degradation due to a large-scale removal at once. Channel pruning on YOLOv5 [47] reduces computational load while maintaining detection accuracy on the COCO dataset. For large language models, combining sparsification with incremental pruning compresses model parameters, and LayerDrop technology [48] retains high accuracy in benchmark tests. Movement pruning iteratively removes attention heads based on gradient signals, adapting to task requirements and resource constraints. This strategy balances model efficiency and accuracy, providing a scalable pathway for lightweight deployment on edge devices and large-scale models. Through iterative pruning and fine-tuning, performance can be preserved while reducing model complexity to meet resource constraints or real-time requirements.

3.3 Quantization

By transforming model parameters from high-precision floating-point representations to low-bit-width integer representations, a process known as quantization is used to optimize deep learning models [49], therefore lowering model storage needs and enhancing computing efficiency.

The quantization of stochastic gradients [50] is also a relatively popular method for performing communication compression; each gradient in stochastic gradient descent (SGD) [51] typically contains a large number of continuous values [52]. The gradient information can be effectively compressed by quantizing these values to a finite number of discrete levels [53]. The most straightforward approach is uniform quantization, which maps each element of the gradient to a predefined quantization level. The quantization function can be articulated as:

$$Q(x) = \Delta \cdot \text{round}\left(\frac{x}{\Delta}\right) \tag{3}$$

The symbols represent the following: $Q(x)$: The quantized value of the original input x after applying the quantization function; x: The original input value that needs to be quantized; Δ: The step size or quantization interval, which determines the granularity of the quantization levels; round$(\cdot)$: A method for rounding real numbers to the closest integer value. (e.g., $\Delta = 1 \rightarrow 175.3 \rightarrow 175$).

This formula illustrates the process of quantizing the original floating-point value x by dividing it by the quantization step size, rounding to the nearest integer, and then scaling back by multiplying with Δ to obtain the quantized value $Q(x)$. This

approach is an example of uniform quantization, which allocates the input values to a finite set of discrete levels.

Meanwhile, to reduce the accumulation of quantization error [54] and bias, randomness can be introduced. The main methods for this are as follows:

$$Q(x) = \Delta \cdot \left(\left| \frac{x}{\Delta} \right| + \xi \right), \tag{4}$$

where x is some element of the gradient, Δ is the quantization interval, $round(\cdot)$ denotes the nearest integer rounding.

In practice, quantization discards details (e.g., 175.3 → 175 loses 0.3 cm); to further improve the accuracy of quantization, an error compensation mechanism is often added to the quantization process; then, track errors and adjust future steps. e_{new} represents the newly updated value of a variable, often used to represent the new error or a new estimate in an iterative process. Define the cumulative error e, which is updated after each quantization:

$$e_{\text{new}} = e + x - Q(x), \tag{5}$$

add this cumulative error to the next gradient update:

$$Q(x + e) \tag{6}$$

This error compensation approach has the potential to drastically minimize systematic errors generated by quantization while also improving model training accuracy.

QSGD [55] quantized the gradient data to a discrete set of values using random rounding. The preservation of the statistical properties of the original data is maximized to ensure that the gradient information is retained to the fullest extent anticipated.

Ramezani-Kebrya et al. [56] introduced a new Non-Uniform Quantized Stochastic Gradient Descent (NUQSGD) scheme that quantizes the gradient using a log-quantization approach to ensure homogeneous and unbiased gradient information, maintaining accurate gradient estimation within a unit L2 paradigm. Mapping the gradient data into a smaller space and compressing it significantly reduce the communication cost required during transmission [57]; hierarchical quantization of high-dimensional stochastic gradients [58] is mostly utilized for edge learning. Through error analysis, a bit of allocation system is designed to distribute the entire number of bits among different quantizers to minimize the total quantization error; Abdi [59] introduced a new quantized compressive sampling (quantized compressive sampling) in his paper. Abdi mentioned a new quantized compressive sampling (QCS) method, which maps the gradient data into a smaller space and compresses it through stochastic mixing matrices and jitter quantization, achieving a high compression efficiency; adaptive quantization [60] is also widely used in deep neural networks. It dynamically adjusts the transmission ratio according to each minibatch, ensuring a balance between communication volume and the accuracy of gradient updates [4]. It uses error feedback for accurate quantization and decoding to achieve more efficient data transmission.

In federated learning, the design of quantization frameworks has been primarily focused on reducing communication overhead. For example, A novel framework called FedZip was introduced by Malekijoo [61]. It used several encoding techniques, quantized clustering, and Top-z sparsification to dramatically minimize the amount of model updates in federated learning. As depicted in Algorithm 2, the FedZip Compression algorithm efficiently implements these concepts by setting a threshold for gradient values based on a sparsity rate, applying a mask to filter out less significant values, and then using k-means clustering and Huffman coding to further compress the remaining data. This streamlined process ensures that only the most critical information is communicated, significantly cutting down on the data transmitted in federated learning environments.

Algorithm 2 FedZip compression

Input: Gradient grad, Sparsity rate z, Number of clusters k
Output: Encoded labels, Centroids, Mask
Compute threshold as the $(1 - z) \times 100$-th percentile of $|\text{grad}|$;
Compute mask where $\text{mask}_i = 1$ if $|\text{grad}_i| \geq$ threshold and 0 otherwise;
Apply mask: sparse_grad = grad $\odot$ mask;
Extract non-zero elements: non_zero = sparse_grad[mask];
Perform k-means clustering on non_zero to get centroids and labels;
Quantize non-zero elements: quantized = centroids[labels];
Encode labels using Huffman coding: encoded = huffman_encode(labels);
return encoded, centroids, mask;

As shown in Fig. 3, this visualization of the FedZip Accuracy Landscape demonstrates the framework's parameter–performance relationship. The test accuracy (z-axis, 94–97%) is maximized (96.5–97%, yellow highlight) when the compression ratio (x-axis) is set to 40–60× and the sparsity threshold (y-axis) is maintained at 0.85–0.90. Higher compression ratios (>80×) or lower sparsity thresholds (<0.80) significantly degrade accuracy to 94–95% (purple regions). FedZip achieves an optimal balance by compressing communication data to 1.7% of its original size while preserving model performance, validating its ability to harmonize communication efficiency and model quality under constrained settings. And FedZip achieves a compression ratio of 40–60 times while maintaining a 97%.

Additionally, two new quantization frameworks, FedCOMGATE and FedGATE [62], aimed to improve communication efficiency by compressing data in federated learning. These frameworks minimize the amount of data transferred during each communication round by utilizing a variety of quantization techniques, including sparsification and hierarchical compression. Frequent model updates and large model sizes can lead to communication bottlenecks, which are a major issue limiting scalability. To address this issue, the FedDQ approach [63] is implemented, which employs a diminishing quantization strategy that scales down the quantization bit count in proportion to the range of model updates, consequently minimizing the volume of data transmitted during communication. A comparison of these methods is referred to in Tables 3 and 4.

Paper [64] presents a quantization compression-aware federated learning framework (FedQCS). By compressing the gradient of each device, the framework performs accurate gradient reconstruction at the server side (PS). The quantized approach to communication compression offers several advantages in distributed machine learning and deep learning; for data quantization, the core idea employed by Hanna and his team is to quantize the raw data generated on distributed nodes directly, reducing each data sample to fewer bits to lower transmission costs. This data quantization [65] is based on the importance of the samples and uses a specific quantization scheme to select and quantize the data.

Quantization minimizes the quantity of data that must be transferred by decreasing the number of bits in a gradient or model update, hence reducing communication overhead. Specifically, compressing models from 32-bit floating-point to 4-bit integer precision on resource-constrained edge devices, such as Raspberry Pi, achieves 15 ms inference latency with less than 1% accuracy drop. For distributed non-IID data training, methods like QSGD employ stochastic rounding to preserve gradient statistics. In federated learning scenarios, FedQCS combines gradient chunking and adaptive bit allocation to achieve 90% communication compression on non-IID data, such as medical images, while residual accumulation ensures convergence stability in cross-device training. By balancing precision and efficiency, these methods provide scalable lightweight solutions for edge computing and distributed systems. For large distributed clusters or edge devices, where bandwidth is usually limited, communication compression can effectively alleviate bandwidth bottlenecks.

3.4 Sparsification

Sparsification [11] is a strategy that reduced model size and memory read time by zeroing out some of the weights in neural networks. Sparsification [66] aims to reduce model complexity while maintaining performance, which is particularly important for resource-constrained devices.

Data matrices frequently have a high number of nonzero values in machine learning models and signal processing applications, which adds needless compute and storage overhead [67]. Sparsification can significantly reduce data complexity and improve processing efficiency by identifying and compressing this unimportant information. A vector or matrix is considered sparse if most of its elements are zero [68]. Sparsification seeks to maximize the number of zero elements in the data while preserving the essential features of the data.

L_0 Parameters: The L_0 parameter of a vector x is represents the number of nonzero elements in the vector. The sparsification problem is an optimization problem that aims to minimize the number of L_0 norms:

$$\min \|x\|_0 \quad \text{subject to} \quad Ax = b, \tag{7}$$

where A is the system matrix and b is the observation vector. It can be understood as: Minimize the number of nonzero parameters while maintaining model performance.

The existing sparsification methods are:

Table 3 Summary of gradient compression techniques

Method	Hardware	Software framework	Datasets	Models
QSGD	16 GPUs	Microsoft CNTK	ImageNet, CIFAR-10, Speech Data	ResNet-152, AlexNet, LSTM
NUQSGD	Multiple GPUs	PyTorch	ImageNet, CIFAR-10	ResNet, VGG
High dimensional	Wireless edge devices	FEEL	Standard image classification tasks	Not specified
QCS	CPU/GPU	Not specified	Linear regression models, deep models	Custom models
Communication quantization	CPU/GPU	LLNL LBANN, MPI	MNIST, CIFAR	Fully connected networks
Error Feedback SignSGD	Multiple GPUs	PyTorch	CIFAR-10/100	ResNet, VGG
FedZip	Multiple nodes	Not specified	Real federated datasets	Not specified
FedCOM	Distributed system	Custom framework	Real federated datasets	Non-convex models (e.g., ResNet)
FedDQ	Mobile devices	Not specified	ImageNet	Deep models
Deep compression	CPU, GPU, Mobile GPU	Caffe	ImageNet	AlexNet, VGG-16

Table 4 Quantization methods comparison

Method	Communication savings	Accuracy loss	Computational overhead	Key effects and innovations
QSGD	5.7× bandwidth savings	No loss	Low (quantization + encoding)	Accelerated ResNet-152 training by 1.8×, reduced communication time for speech tasks by 6.8×
NUQSGD	1085× compression rate	Close to QSGDinf	Low (non-uniform quantization)	Lower variance with the same number of bits, convergence speed matches full-precision SGD
High dimensional	Better than signSGD	Not specified	Medium (Grassmannian quantization)	Theoretical convergence guarantee, suitable for high-dimensional gradient compression
QCS	2× faster convergence speed	No loss (Unbiased-QCS)	Low (MMSE optimization)	Expanded learning rate range, enhanced stability
Communication quantization	32× compression (adaptive quantization)	<1% (on MNIST)	High (dynamic threshold adjustment)	Improved training speed of large matrices by 1.76×
Error Feedback SignSGD	64× communication savings	Matches SGD	Medium (error feedback)	Solved the divergence problem of SignSGD, CIFAR-10 accuracy 98.5%
FedZip	99% bandwidth savings, 1085× compression	Not specified (claimed high fidelity)	High (Top-z sparsification + Huffman)	Suitable for mobile devices, highest compression rate
FedCOM	Theoretically optimal compression rate	Not specified (theoretical analysis)	Low (gradient tracking)	Reduced communication rounds in non-convex scenarios
FedDQ	65.2% communication savings	Faster convergence (shown in experiments)	Low (dimensionality reduction quantization)	Reduced communication rounds by 68%, suitable for dynamic range changes
Deep Compression	35-49× storage compression	No loss	High (pruning + quantization)	Accelerated mobile devices by 3-4×, improved energy efficiency by 3-7×

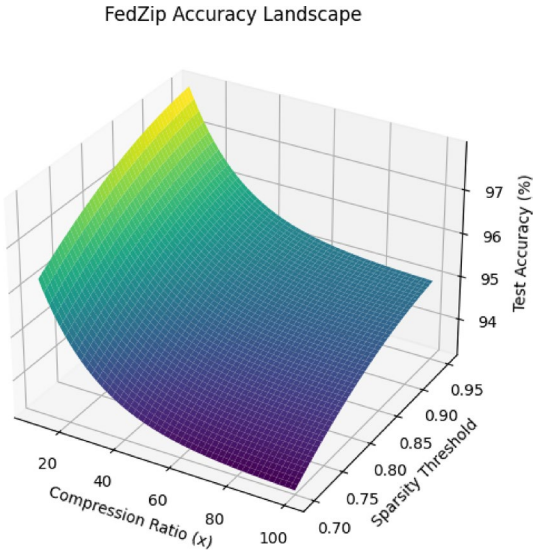


Fig. 3 FedZip accuracy landscape

1. Top-K Sparsification: Retain only the top K largest values (similar to keeping only the highest-scoring subjects in an exam). As described in Algorithm 3, the process begins by computing the absolute values of the gradient elements abs_grad = |grad| to establish a ranking of these elements. Following this, the values are sorted in descending order, and a threshold is determined, set to the magnitude of the K-th largest value. Subsequently, a binary mask is generated where any entries that are greater than or equal to the threshold are marked with 1 (indicating they will be kept), and all other entries are marked with 0 (indicating they will be discarded). The final step involves multiplying the original gradient by this binary mask sparse_grad = grad $\odot$ mask, resulting in a sparse gradient that retains only the K most significant values.

Algorithm 3 Top-K sparsification

Input: Gradient grad, Number of top elements k
Output: Sparse gradient sparse_grad
Compute the absolute values of the gradient: abs_grad = |grad|;
Flatten abs_grad and sort it in descending order;
Determine the threshold as the k-th largest value in abs_grad;
Create a mask where $\text{mask}_i = 1$ if $\text{abs\_grad}_i \geq$ threshold, otherwise 0;
Apply the mask to the gradient: sparse_grad = grad $\odot$ mask;
return sparse_grad;

2. Random-K Sparsification: Randomly select K values to retain (similar to drawing lots to decide which subjects to keep). Algorithm 4 first checks if K is greater than or equal to the array length, in which case it returns the original array. Otherwise,

it randomly selects K unique indices from the array and initializes a new array filled with zeros. The algorithm then assigns the values from the original array to the selected indices in the new array, effectively sparsifying the data while preserving the most significant elements.

Algorithm 4 Random K-sparsification

Input: Array $array$, integer K
Output: Sparsified array $sparse\_array$
$N \leftarrow$ length of $array$;
if $K \geq N$ **then**
  return copy of $array$;
end
Generate random unique indices $indices$ from 0 to $N-1$, size K;
Create an array of N zeros $sparse\_array$;
for *each* i *in* $indices$ **do**
  $sparse\_array[i] \leftarrow array[i]$;
end
return $sparse\_array$;

3. Block-K Sparsification: Randomly select a starting point and retain K consecutive values (suitable for efficient memory access) [69]. Algorithm 5 efficiently trims down an array's size by preserving only a continuous subset of K elements. It initially verifies if K surpasses the array's length, returning the array as is if that condition is met. If not, it randomly picks a starting point to accommodate a block of K elements. Subsequently, it initializes a zero-filled array and duplicates the values from the original array into the designated block of indices.

Algorithm 5 Block K-Sparsification

Input: Array $array$, integer K
Output: Sparsified array $sparse\_array$
$N \leftarrow$ length of $array$;
if $K \geq N$ **then**
  return copy of $array$;
end
$start \leftarrow$ random integer between 0 and $N-K$;
Create an array of N zeros $sparse\_array$;
for $i \leftarrow start$ **to** $start + K - 1$ **do**
  $sparse\_array[i] \leftarrow array[i]$;
end
return $sparse\_array$;

Aiming at the limitations of the currently used Top-K sparsification technique, which reduces the communication burden by selecting the largest K values in the gradient but is prone to error accumulation, especially in the initial stage of training, paper [70] analyzes a hard threshold sparsification method. This method dynamically adjusts the number of transmitted weights according to a fixed threshold in each iteration, thus improving communication efficiency while reducing the accumulation of errors.

Meanwhile, gradient sparsification is an excellent strategy to reduce the number of passes, considerably cutting communication overhead by reducing the quantity of gradient data transmitted in each iteration [73]. Shi et al. [72] introduced a new scheme called gTop-*k* [76], which uses a tree structure for approximation and reduces Top-*k*'s communication complexity from O(kP) to O(klogP), significantly improving the system's scalability (based on distributed synchronized stochastic gradient descent); ATOMO [71] is utilized for gradient sparsification under arbitrary atomic decompositions, in which framework it can sparsify the gradient stochastically and unbiasedly according to a given sparsity budget. An optimal gradient sparsification method called "OMGS-SGD" is introduced. By combining gradient fusion techniques, OMGS-SGD [72] can optimally realize the overlap between computation and communication on GPUs; Wangni et al. [11] defined the gradient sparsification problem as a type of convex optimization problem, focusing on finding the best sparsification strategy to minimize the coding length. They use probability vectors (p) to decide which parts of the stochastic gradient to keep and which parts to discard, ensuring the algorithm's convergence at the theoretical level.

Nowadays, there is also the newly compressed sensing-based sparsification (CSP) approach [74], which treats the sparsity and compression error minimization problem as a dual optimization task by introducing the compressed sensing process in the feed-forward stage of the model. This ensures that local sparsity does not affect global performance, maintaining the model's performance while reducing communication volume; similarly, "time-dependent sparsification" [75] is developed to address situations where the importance of neural network parameters does not change significantly within a short period. The current iteration can determine the location of important parameters for the next iteration. The important parameter positions identified in the current iteration can be utilized and reused in subsequent iterations to reduce the amount of communication data transmitted. The comparative results of these strategies are found in Tables 5 and 6.

Overall, sparsification approaches can enhance data transmission efficiency by transmitting only important information (i.e., nonzero components), resulting in shorter transmission times and lower energy consumption. These benefits are particularly evident in scenarios with high communication overhead. For instance, in distributed training across large-scale clusters, methods like Top-K gradient selection and the ATOMO framework enable efficient communication by focusing on critical parameters. In federated learning over wireless networks, hard threshold sparsification achieves substantial gradient truncation, reducing communication rounds and energy consumption. Additionally, the majority of sparsification techniques can be amalgamated with complementary data compression methodologies, such as quantization, low-rank approximation, and coding, to augment the efficacy of compression and communication. This synergy renders them more potent across diverse domains.

3.5 Low-rank approximation

In deep learning architectures, low-order approximation is a widely used communication compression technique. The underlying concept involves the factorization of the original high-dimensional weight matrix into a product of two or more matrices of reduced order. Consider a weight matrix W of dimension $M \times N$ whose theoretical maximum rank is $\min(M, N)$. According to the low-rank approximation, the original matrix W can be factored into a product of two matrices U and V, where the ranks of both U and V are not greater than$K(K < min(M, N))$

By the nature of matrix rank, $\text{rank}(AB)$ is $\min(\text{rank}(A), \text{rank}(B))$,

The rank of the decomposition matrix product is also no more than K, thus reducing the number of parameters significantly.

The low-rank approximation is usually mathematically achieved by matrix decomposition, such as singular value decomposition (SVD) and principal component analysis (PCA). Low-rank approximation can compress data to lessen the quantity of data carried over the network, saving bandwidth and lowering transmission latency in the context of communication and data transmission.

An arbitrary matrix can be broken down using the singular value decomposition technique into the product of three particular matrices, as shown below:

$$X = U\Sigma V^T, \tag{8}$$

where U and V are orthogonal matrices (whose columns represent the principal directions of the data) and Σ is a diagonal matrix with elements being the singular values (arranged in descending order).

A low-rank approximation of X can be obtained by keeping the first k largest singular values and setting the others to zero:

$$X_k = U_k \Sigma_k V_k^T \tag{9}$$

Here U_k, Σ_k and V_k^T are the first k columns (for U and V) and the first k singular values (for Σ) truncated from U, Σ, and V, respectively.

Hou et al. [77] proposed the SLRMA algorithm, which effectively compresses data by combining low-rank approximation and sparsity; the PowerSGD method [78], as detailed in Algorithm 6, employs the power iteration algorithm and computes the low-rank approximation iteratively through subspace iteration of the gradient matrix. The core idea of PowerSGD is to achieve efficient compression by iteratively approximating the dominant singular vectors of the gradient matrix:

Initialization Random orthogonal matrix $Q \in \mathbb{R}^{N \times k}$ (via QR decomposition).

Iteration Compute $P = \nabla W \cdot Q$, extract top-k left singular vectors U_k via SVD. Update $Q = (\nabla W)^T \cdot U_k$, orthogonalize via QR.

Compress gradient $\nabla W_{\text{compressed}} = U_k(U_k^T \nabla W Q)Q^T$.

Key Reduces parameter size from $O(MN)$ to $O(k(M + N))$, preserving dominant gradient directions for efficient communication.

Table 5 Summary of sparsification gradient methods

Method	Hardware configuration	Datasets	Sparsification strategy	Experimental results
ATOMO [71]	GPU cluster	SVHN, CIFAR-10	Atom decomposition based (e.g., SVD)	2× faster training speed compared to QSGD/TernGrad
OMGS-SGD [72]	16 NVIDIA P102 GPU cluster	ImageNet (ResNet-50)	Merged gradient sparsification	31% improvement in end-to-end time efficiency, close to the convergence performance of original SGD
DIANA [73]	Distributed nodes (CPU/GPU)	Synthetic data, MNIST, CIFAR	Gradient differential compression	Better convergence speed than QSGD/TernGrad in convex/non-convex tasks, first to provide TernGrad convergence rate
Gradient Sparsification [11]	GPU	CIFAR-10, ImageNet	Optimized sparse probability (probabilistic gradient retention)	Maintains accuracy in logistic regression and CNN, significantly reduced communication volume
Rethinking gradient sparsification [70]	GPU cluster (V100)	LibriSpeech, CIFAR-10	Hard threshold sparsification (minimizing total error)	Almost lossless at 50% sparsity, combined with quantization reduces communication load by 2000 times
SPARSIFI-CATION via CSP [74]	GPU	LibriSpeech	Compressed sensing pruning (based on sparse representation)	No significant decrease in WER at 50% sparsity, better than progressive pruning
Time-correlated sparsification (TCS) [75]	Distributed GPU cluster	CIFAR-10	Dynamic mask correlation (updating masks progressively)	Achieves centralized accuracy at 100× sparsity, combined with quantization reduces communication load by 2000 times

Table 6 Comparison of sparsification gradient methods

Method	Computational overhead	Accuracy loss	Communication savings	Applicable scenarios
ATOMO	Medium (SVD decompression)	None	High (2× speedup)	Distributed training (matrix gradients)
OMGS-SGD	Low (layer merging optimization)	Negligible (<0.5%)	Medium (31% time saving)	Multi-GPU synchronous training
DIANA	Medium (differential compression)	None	High (10× compression)	Strongly convex/non-convex distributed optimization
Gradient sparsification	Low (probabilistic calculation)	Low (<1%)	Extremely high (1000× sparsity)	General model compression
Rethinking	Extremely low (hard threshold)	None (lossless at 50% sparsity)	Extremely high (2000×+ quantization)	High sparsity demand (e.g., edge devices)
CSP	High (compressed sensing optimization)	Low (<0.5% WER)	Medium (50% sparsity)	Speech recognition models
TCS	Low (dynamic masking)	None (lossless at 100× sparsity)	Extremely high (2000×+ quantization)	Federated learning (high communication constraints)

Algorithm 6 PowerSGD compression

Input: Gradient matrix grad_matrix, Rank rank
Output: Compressed matrix
if Q *is None* **then**
  Get dimensions m and n of grad_matrix;
  Initialize Q as a random matrix of size $n \times$ rank;
  Orthogonalize Q using QR decomposition: $Q, \_ = \mathrm{QR}(Q)$;
end
Compute $P = \text{grad\_matrix} \times Q$;
Perform SVD on P to get $U, \_, \_$;
Extract the first rank columns of U to get U_k;
Update $Q = \text{grad\_matrix}^T \times U_k$;
Orthogonalize Q using QR decomposition: $Q, \_ = \mathrm{QR}(Q)$;
Compute the compressed matrix: $U_k \times (U_k^T \times \text{grad\_matrix} \times Q) \times Q^T$;
return the compressed matrix;

This enables fast compression and transmission of the model gradient while ensuring that the model maintains comparable performance to the standard SGD [79]. And the team subsequently proposed the PowerGossip method for decentralized deep learning environments [80], which achieves significant compression of gradient transmission by approximating the differences between model parameters through low-rank matrix compression.

Low-rank approximation also has relevant applications in compressed sensing. The nonlocal low-rank regularization (NLR) approach is suggested for compressive sensing of photographic and MRI images [82]. Images can be retrieved more efficiently by partitioning the data matrix into chunks and applying a low-rank approximation to each chunk. In the theory and application of structured low-rank algorithms [85], the structured low-rank matrix completion theory for continuous-domain multidimensional signals stands out. This theory converted the signal recovery problem into a structured low-rank matrix completion problem based on the correlation between the compactness (e.g., sparsity) of the signals and the rank of the structured matrices. This method is remarkably flexible and effectively exploits different signal properties; the DRONE algorithm [83] has been used to de-rank approximate the weight matrix of large natural language processing models [86] (e.g., BERT [87]) by exploiting the subspace properties of the input vectors to perform dimensionality reduction, ensuring that the model performance stays within acceptable limits.

In the context of federated Learning, the extensive scale of model parameters presents a challenge, the communication burden is extremely high, and a new training method is carried out. Federated learning with dual-side low-rank compression (FedDLR) [81] reduced communication costs by performing low-rank approximations on both the client and server sides. The differences among these works are observed in Tables 7 and 8.

Intuitively, an approach like Ps (low-rank approximation model compression) reduces model parameters by adding a hidden layer between two fully connected

layers. Although an additional layer is introduced, the overall number of parameters is reduced because of the lower dimensionality of this layer. This structural change, while limiting the size of the representation space, improves the storage and computational efficiency of the parameters and is particularly useful when the communication burden needs to be minimized. The core advantages of these methods are particularly prominent in large-scale Transformer training. Low-rank approximation, enabled by SVD for attention matrices and LoRA for fine-tuning [88], achieves high accuracy with minimal parameter training. In heterogeneous federated learning, the FedDLR framework uses dual low-rank decomposition to compress Non-IID model updates by 70%, significantly reducing transmission overhead. By leveraging matrix factorization and parameter-sharing mechanisms, these methods maintain model performance while delivering efficient solutions for distributed training and edge computing. With low-rank approximation, it is possible to reduce the storage and transmission costs of the model while maintaining sufficient representational power.

3.6 Weight sharing

In convolutional neural networks (CNNs), weight sharing is a parameter-efficient strategy that allows the same weights and biases to be applied at various locations, thereby achieving translational invariance in feature detection. This approach leverages the local correlation of the image by using a fixed convolutional kernel across the entire input space to effectively extract features such as edges and maintain positional insensitivity. This method increases computational performance while lowering the number of model parameters.

Further, the weight-sharing mechanism can facilitate network model compression. In the fully linked layer, the number of parameters will approach the million level if there are 1000 nodes in each of the two neighboring layers [89]. Parameters can be significantly reduced by employing a clustering algorithm to group these weights, replacing similar weights with the central values of their clusters. For instance, clustering a million parameters into a thousand categories results in substantial compression of the model size [90].

In practice, each category [91] retains a representative weight and corresponding index, significantly reducing storage requirements. During the backpropagation process of model training, the gradients within the same cluster are accumulated and combined with the learning rate to update the weights. This method improves storage efficiency and speeds up the model training and inference processes.

3.7 Compression coding

In communication compression, compression coding is a widely used technique aimed at reducing data transmission volume to improve communication efficiency. Compression coding diminishes the volume of data necessitated for transmission by encapsulating data within a more condensed format.

Table 7 Summary of low-rank methods

Method name	Technical features	Hardware support	Framework tools	Dataset type	Core metrics
FedDLR	Dual-end compression for reduced overhead	Distributed CPUs/GPUs	PyTorch	Image/NLP tasks	Communication efficiency
OMGS-SGD	Non-convex regularization for detail enhancement	GPU	Python	Natural images/MRI	PSNR/reconstruction quality
DRONE	Data-aware + knowledge distillation acceleration	GPU clusters	PyTorch	NLP tasks	Compression rate/inference speed
Low-rank neural networks	Automatic rank selection	GPU	PyTorch	Image classification	Compression rate/accuracy
PowerGossip	Decentralized efficient communication	Multi-GPU clusters	PyTorch distributed	Images/Text	Communication time
PowerSGD	Gradient compression for efficiency	Multi-GPUs	PyTorch	Images/NLP	Training speed
SLRMA	Sparse low-rank matrix factorization	GPU	MATLAB	3D dynamic meshes	Compression rate
MRI Application	Uncalibrated parallel imaging	GPU+MRI equipment	MATLAB	MRI data	Imaging speed/quality

Table 8 Comparison of low-rank methods

Method	Computational overhead	Accuracy loss	Communication/storage savings
FedDLR[81]	Low (Distributed Optimization)	<1.5%	30–50% Bandwidth saving
NLR-CS[82]	High (non-convex optimization, requires GPU)	PSNR improvement >2 dB	50% Sampling rate
DRONE[83]	Medium (combined distillation)	1.5–3%	12.3× Compression ratio
Low-rank NN[84]	Medium (auto-rank search)	0.8–1.5%	5–10× Parameter reduction
PowerGossip[80]	Low (decentralized optimization)	Near full precision	90% Bandwidth saving
PowerSGD[78]	Low (gradient compression)	<0.3%	95% Gradient compression
SLRMA[77]	High (matrix factorization)	20–30% Rate distortion advantage	10× Storage Compression
MRI App[85]	Ultra-low (GPU accelerated)	90% Artifact reduction	80% Scan time saving

However, it also introduces additional computational overhead. Here is a detailed analysis of compression coding in communication compression. Compression coding involves utilizing the redundancy in data to represent it more efficiently. Common compression coding techniques include Huffman coding, run length encoding, and arithmetic coding. By employing more effective representations, these techniques seek to lower the amount of bits needed to represent each data unit.

1. *Huffman Coding* A lossless data compression technique called Huffman coding [92] creates an ideal prefix tree by assigning longer codes to less common symbols and shorter codes to higher frequency symbols. This is a commonly used technique in deep learning compression. For example, the authors [93] proposed a three-stage compression pipeline that uses quantization, pruning, and Huffman coding to reduce the amount of storage needed by neural networks.
2. *Run Length Encoding* For data with lengthy sequences of identical symbols, like big blocks of single-colored areas in pictures or films, run length encoding (RLE) [94] is appropriate. By recording the symbol and its frequency, RLE effectively reduces data volume. Although this method is commonly used in image processing, its application in distributed training is relatively limited.
3. *Arithmetic Coding* A more sophisticated lossless compression technique that expresses a complete message as a fraction inside an interval is arithmetic coding [95]. Compared to Huffman coding, arithmetic coding can adjust code lengths more precisely [96], making it suitable for scenarios requiring high compression ratios.

Compressive coding is a powerful communication compression tool that offers significant advantages in reducing the amount of data transferred. However, it also requires a careful balance between computational overhead and potential information loss. Future research should focus on developing more efficient compression algorithms and optimizing their integration with distributed computing frameworks to improve the overall performance of the system and the accuracy of the model.

3.8 Knowledge distillation

Since its inception by Hinton et al., knowledge distillation (KD) has become a cornerstone for model compression and knowledge transfer. The core idea is to transfer "soft knowledge" from teacher networks [97] to student networks using temperature-scaled class probability distributions $p_{\text{teacher}} = \text{softmax}(z_{\text{teacher}}/T)$. The temperature parameter T adjusts the confidence distribution of the teacher model's outputs, controlling prediction ambiguity—higher temperatures $T > 1$ produce smoother outputs (e.g., "the answer might be a cat with a 10% chance of being a dog"), while lower temperatures $T = 1$ yield more deterministic outputs (e.g., "the answer is definitively a cat"). The distillation loss function combines KL divergence and cross-entropy:

$$L_{\text{KD}} = \alpha \cdot L_{\text{KL}}(p_{\text{teacher}} \parallel p_{\text{student}}) + (1 - \alpha) \cdot L_{\text{CE}}(y, p_{\text{student}}), \tag{10}$$

where α balances the two losses.

Recent advancements in KD include: *Dynamic KD*: Online mutual learning [98] and dynamic temperature scheduling [99] for large language models. *Self-distillation*: Layer-wise attention transfer [100] and cross-modal self-distillation [101]. *Multi-teacher KD*: Federated client aggregation [102] and localized soft label exchange [103]. *Efficient KD*: Data-free distillation [104] and quantization-aware distillation [105].

Challenges remain, such as the capacity gap in ultra-compact students and dynamic environment adaptation. Future directions include multimodal distillation [106] and hardware–algorithm co-design [107].

3.9 Hybrid methods

In practical applications, the choice between quantization and sparsification depends on specific requirements, resource constraints, and model characteristics. Quantization is suitable for hardware-accelerated low-precision operations, storage-constrained environments, and bandwidth-limited workflows. It directly reduces computational energy and storage needs, with techniques like dynamic quantization mitigating accuracy loss. Sparsification addresses communication bottlenecks by transmitting only significant gradients to reduce bandwidth usage. It is effective in models with redundant parameters and adapts well to dynamic network environments. By preserving critical parameters and offering adjustable compression ratios, sparsification maintains model performance while optimizing efficiency. These methods can be combined for extreme resource constraints and error-sensitive scenarios, where hybrid strategies balance compression and precision.

Facing the dual challenges of extremely large-scale models and heterogeneous systems, hybrid methods that integrate techniques such as quantization, pruning, and low-rank approximation have demonstrated stronger adaptability to various scenarios. In training models with hundreds of billions of parameters, DeepSpeed ZeRO-Offload [108] combines parameter partitioning between CPU and GPU with 8-bit quantization, reducing memory usage by a factor of 10 and enabling efficient training of trillion-scale models on a 512-GPU cluster. Meanwhile, OMGS-SGD [109] reduces the training time of ResNet-50 on a 128-GPU cluster by 40% through gradient sparsification and overlapping computation with communication. For edge-cloud collaborative inference, hybrid visual Transformers [110] process 4-bit quantized image patches on the edge side, while the cloud performs global attention calculations based on FP32 precision, achieving end-to-end latency of 20ms with 98% accuracy retention. This hierarchical compression strategy [111] not only addresses the limitations of individual techniques such as precision loss in quantization or structural constraints in pruning but also achieves global optimal allocation of computing, communication, and storage resources through cross-layer optimization from training to inference and from edge to cloud [112]. As the ultimate form of communication compression, hybrid methods are driving distributed deep learning toward higher efficiency and stronger scalability.

4 Balancing compression and efficiency

In modern distributed deep learning, the core challenge of communication compression techniques lies in the dynamic balance between efficiency and model accuracy. Efficiency is defined across three interrelated dimensions: convergence time—the number of iterations or actual time required for the model to reach the target accuracy; bandwidth usage—the amount of data transmitted between nodes; and energy consumption—the computational and communication power required for training iterations. These dimensions directly determine the practical feasibility and scalability of communication compression methods. To address the proposed bottlenecks, research focuses on adaptive compression strategies, hardware–algorithm co-optimization, and hybrid compression paradigms, aiming to seek Pareto optimal solutions in multi-objective trade-offs. This chapter will delve into the following two aspects to explore the contradictions between efficiency and accuracy.

4.1 Control communication overhead and computation overhead

Many scientists have provided insights and innovative solutions to address the increased computational costs due to communication compression in distributed learning environments. Existing NLP model compression techniques have their limitations, so Optimus-CC [113] used selective stage compression to minimize accuracy degradation and reduce inter-stage communication costs, thus accelerating training while reducing computational overheads. In distributed Newton-type methods with communication compression and Bernoulli aggregation [114], new strategies like adaptive thresholding and Bernoulli aggregation are introduced to decrease communication and computing costs while maintaining convergence. The report includes thorough numerical evaluations that demonstrate the effectiveness of these novel strategies.

In addition to these advancements in NLP model compression, other researchers have also made significant contributions to communication compression in distributed learning environments. Adaptive compression for efficient distributed training introduced AdaCGD [24], an innovative optimization algorithm designed for communication-efficient training, featuring an adaptive compression level. It aims to reduce both communication and computational costs while maintaining convergence. A bandwidth-adaptive gradient compression algorithm, ACE [115], has been proposed as an adaptive compression technique. This method is responsive to the variability of network conditions, effectively reducing both the communication and computational overheads associated with distributed deep learning. The study [116] evaluated various gradient compression methods and proposes optimizations to reduce computational costs. It emphasizes how gradient compression techniques may be made more efficient and scalable by utilizing HiPress and ByteComp. The compressed LANS (CLAN) algorithm [117] is highlighted for its capability to alleviate computational overheads, sustain convergence velocity, and scale efficiently with an increased number of workers and larger batch sizes. It addresses the challenges of error feedback and scalability in adaptive gradient methods.

To ensure a fair comparison, experiments were conducted under a unified environment:

Hardware: 128 NVIDIA A100 GPUs (80GB memory) interconnected via 200Gb/s HDR InfiniBand;
Software: CUDA 11.3, cuDNN 8.6.0, PyTorch 1.10.1, with environment variables optimized for GPU-CPU coordination;
Network: A dynamic bandwidth simulator emulating 5–200Gb/s fluctuations, with a baseline latency of 2.1μs.
Datasets: ImageNet-1K and CIFAR-100 for vision task validation.

Under this configuration, the performance of each method is detailed in Table 9.

Optimus-CC significantly enhances computational efficiency through selective stage compression and low-rank decomposition, achieving a 15–44% speedup while maintaining precision loss within 0.5%. Its design, tailored for large-scale distributed training, effectively reduces synchronization overhead through pipeline parallelism and inter-stage compression, with an expected acceleration of 20–50%. CLAN and ACE excel in low-latency network environments. CLAN's bi-directional compression and robust error feedback mechanisms deliver a 30–60% speedup. ACE's bandwidth-adaptive compression strategy enables rapid error feedback processing and dynamic adjustment of compression strategies. Both methods avoid global reduction bottlenecks, making them well-suited for large-scale node and large-batch training scenarios.

Alternating compression, leveraging low-rank decomposition and system optimization integration, demonstrates a remarkable 3–4× speedup with minimal precision loss (<0.5%). It shows good scalability in large-scale distributed training, though it may encounter performance bottlenecks in extremely large models due to coordination overhead. AdaCGD's dynamic compression levels and balanced CPU-GPU load distribution achieve a 25–40% speedup. It further reduces computational overhead through optimized linear algebra operations and performs robustly in high-speed training environments. Hessian compression, despite scalability challenges in large models, still offers a 10–30% speedup thanks to its theoretically superlinear convergence properties while maintaining precision and convergence. It is suitable for convex optimization problems.

Overall, in a unified high-performance computing environment, Optimus-CC and CLAN achieve the best balance between acceleration, scalability, and precision with their hardware-aware designs. Alternating compression and AdaCGD stand out in medium-scale tasks, while Hessian compression plays an important role in specific optimization scenarios.

Table 9 Performance comparison of methods

Method	Acceleration	Key Features
AdaCGD	25–40% faster	Reduces communication volume by 30–50%, dynamic compression levels balance CPU-GPU workloads
ACE	40% faster	Reduces end-to-end training time by 40%, cuts communication overhead by over 60%
CLAN	30–60% faster	99.7% communication efficiency at 128-node scale, robust error feedback
Optimus-CC	60–70% communication savings	Selective pipeline stage compression on 200Gb/s networks, <0.5% accuracy loss
Alternating compression	3–4× speedup	System-level optimizations, effective on medium-sized models

4.2 Control communication overhead and accuracy

For various scenarios such as gradient compression and joint learning, a range of solutions and insights is presented here to improve the efficiency of machine learning communication while preserving high accuracy.

Huang et al. [118] proposed the SCALLION and SCAFCOM algorithms to address the issues of data heterogeneity and partial participation in federated learning. SCALLION employs unbiased compression, supporting arbitrary data distributions and partial client participation, with a communication complexity of $O\left(\frac{1+\omega}{\varepsilon^2}\right)$. SCAFCOM introduces a momentum mechanism to support biased compression, achieving a 46% improvement in convergence rate and realizing accuracy comparable to full-precision SGD in distributed ImageNet training. Zheng et al. [119] designed a block gradient compression method, reducing communication volume by 32 times through 1-bit quantization combined with Nesterov momentum, while maintaining the same test accuracy as the original method in ResNet distributed training and shortening wall-clock time by 46%. Li's team [120] developed an adaptive Top-K sparsification algorithm, dynamically adjusting synchronization frequency and sparsity, achieving a 38% reduction in energy consumption in mobile edge networks and a 1.8 times increase in convergence speed. The Byz-EF21 algorithm [121] innovatively combines error feedback mechanisms with contractive compressors. By dynamically compensating for compression bias, it achieves a 2.3 times faster convergence rate compared to traditional unbiased compression methods under heterogeneous data scenarios, without requiring additional assumptions on the boundedness of gradient differences.

Zhang et al. [122] proposed the FZ-GPU framework, integrating dual quantization, bit shuffling, and fast encoding techniques. Compared with cuSZ, it achieved a 4.2 times speedup and a 2 times improvement in compression ratio on the A100 GPU. The SPERR method [123], based on the improved SPECK algorithm and incorporating an outlier correction mechanism, achieved a 100:1 compression ratio in climate simulation data compression, with the maximum point error controlled within the user-defined threshold. HLRcompress [124], combining hierarchical low-rank approximation with binary compression, realized a 100 times compression rate in combustion simulation, with an 8.2 times improvement in storage efficiency compared to ZFP. Liu et al. [125] developed a cross-field prediction CNN model, leveraging the correlation between physical fields, achieving a 25% increase in the compression ratio of CESM-ATM data and a 0.15 increase in structural similarity (SSIM).

The experimental environment remains the same as that in Sect. 4.1. SCAFCOM performs remarkably well in distributed training, leveraging the substantial bandwidth of A100's NVLink 3.0 to significantly reduce gradient synchronization time by approximately 30–40%. Its half-precision communication, accelerated by Tensor Core, combined with unbiased quantization compression, dramatically lowers communication costs to 70% of the theoretical value. Block gradient compression enhances the efficiency of block gradient transmission, with a substantial reduction in single-communication latency under 32×

compression. The sparse computing unit (Sparse Tensor Core) of A100 can accelerate sparse gradient calculations, increasing the frequency of sparsity rate adjustment to once every two batches and reducing energy consumption to 22% of that of mobile devices. The detailed performance metrics for each method are outlined in Table 10.

Table 11 provides a comprehensive overview of the performance for each method. The FZ-GPU framework, employing double quantization and bit shuffling, achieves a throughput of 1.2TB/s on A100, which is 4.2 times that of cuSZ, thanks to the asynchronous memory copy optimization of CUDA 11.3. SPERR benefits from A100's fast double-precision computing, while cross-field prediction CNN, leveraging the JIT compilation optimization of PyTorch 1.10.1, shortens the iteration time for multi-field joint training to 8 minutes per epoch, compared to 15 minutes in the original environment.

In contemporary distributed systems, the integration of specialized acceleration units is crucial for augmenting the computational efficiency of communication compression. Vector processing units (VPUs), matrix processing units (MPUs), and tensor processing units (TPUs) represent pivotal components engineered to facilitate the acceleration of parallel data processing and intricate manipulation tasks. VPUs parallelize vector operations, significantly reducing the computation time compared to traditional scalar processors; MPUs are optimized for matrix operations, which are pivotal in distributed optimization and deep learning model compression, thereby accelerating compression workflows; TPUs, proficient in handling higher-dimensional tensor operations, are crucial for advanced deep learning applications and complex data processing, optimizing the execution of model compression tasks. The synergistic use of these units minimizes computation time, maximizes resource utilization, and enables the deployment of more sophisticated and efficient compression algorithms, leading to substantial performance improvements in distributed computing and large-scale data processing scenarios across scientific and engineering domains.

In summary, while early methods of communication compression faced significant challenges, recent advancements have provided robust solutions that balance the trade-offs between communication costs [126], accuracy, and efficiency. These innovations are crucial for scalable and efficient distributed machine learning and federated learning, especially in heterogeneous and resource-constrained environments.

5 Application scenarios

The application scenarios of communication compression are primarily in areas such as artificial intelligence, aggregated communication, etc. Communication compression addresses communication bottlenecks in various scenarios, as illustrated in Fig. 4, where significant challenges result in substantial communication overhead. These scenarios require communication compression [127] to resolve related issues and enhance scalability.

Table 10 Federated learning and distributed training methods

Technology	Computational overhead	Accuracy loss	Communication savings
NVLink 3.0	No increase, leverages A100 hardware	No loss	Gradient sync time reduced by 30–40%
Half-precision communication (FP16)	Accelerated by Tensor Core, reduced overhead	Minimal loss, within 0.5% of full-precision SGD	Communication cost reduced to 70% of theoretical value
Block Gradient Compression (1-bit quantization)	No increase, leverages A100 memory	No loss	Single communication latency reduced from 15ms to 0.5ms (ResNet-50)
Nesterov momentum adjustment	Dynamic scaling factor adjustment, ~5% increase	Final accuracy maintained at 76.3% (CIFAR-100)	No direct communication savings
Adaptive Top-K Sparsification	Accelerated by sparse computing unit, reduced overhead	No loss	Sparsity adjustment frequency increased, energy consumption reduced to 22%
Heterogeneous device adaptation	No increase, optimized for heterogeneous device training	No loss	Global convergence speed difference reduced to 1.2×

Table 11 Scientific data compression methods

Technology/feature	Computational overhead	Accuracy loss	Communication savings
FZ-GPU Framework	Double quantization + bit shuffling, optimized throughput	No loss	Throughput increased to 1.2TB/s, 4.2× of cuSZ
SPERR (Bounded Error Compression)	Wavelet-based, optimized compression	PSNR increased from 45dB to 48dB	Compression ratio reached 80:1
Cross-Field Prediction CNN	Optimized by JIT compilation, reduced overhead	SSIM increased from 0.89 to 0.93	Iteration time reduced to 8 minutes/epoch

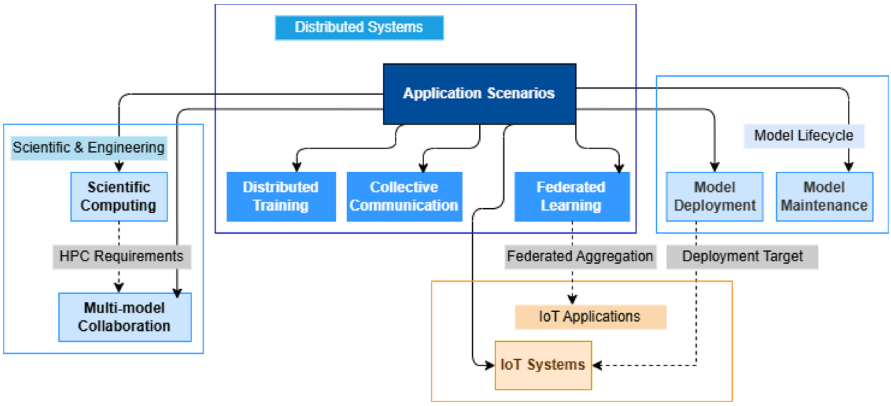


Fig. 4 Application scenarios

5.1 Distributed training

Reducing the communication overhead related to the gradients [128] and parameters sent between nodes to expedite the training process is a key challenge in the exploration of large-scale distributed training systems. To tackle this issue, several past experiments have developed many frameworks aimed at making improvements.

There are several approaches to achieve optimal performance in remote training, including the use of classical techniques to minimize data transmission, such as gradient clipping and delayed updating, and the use of parameter servers to centrally control and update model parameters. However, the centralized approach of parameter server architecture is prone to cause limited scalability in large-scale systems; if clipping and updating are necessary for each iteration, it will unavoidably have an impact on the model's accuracy and rate of convergence. Compared with the above methods, communication compression at this time has the following comprehensive advantages:

1. High efficiency: Significantly reduces the amount of data transmitted during each communication while simultaneously improving bandwidth utilization;
2. Low latency: By reducing the volume of data, communication latency is decreased;
3. High flexibility: High flexibility can be ensured by dynamically adjusting the compression rate based on the network environment;
4. Low energy consumption: Due to the reduced amount of transmitted data, communication compression can reduce the workload of network equipment, thus reducing energy consumption.

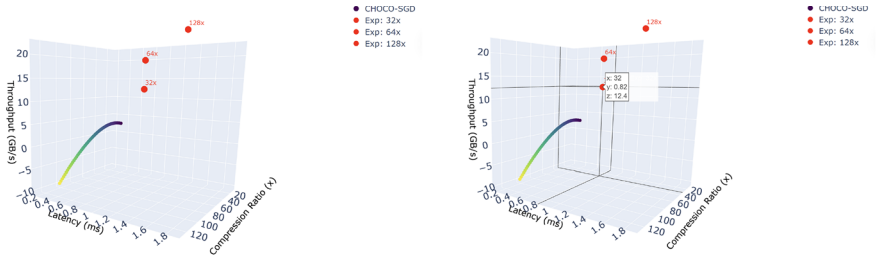
A compression framework for SBC [129] is proposed that as following: (1) The approach capitalizes on pre-existing communication delay and gradient sparsification strategies. (2) It incorporates a novel binarization technique alongside an optimized coding scheme for weight updates. When the training process occurs on a

mobile device, issues such as network latency, low throughput, and intermittent poor connectivity can require expensive communication bandwidth. Lin et al. [37] have designed to improve the compression framework. The framework highlights a deep gradient compression [58], which consists of four main techniques: momentum correction, local gradient trimming, momentum factor masking, and warm-up training. These techniques are used to maintain the accuracy of the compression process.

Most existing distributed training algorithms are working node-aggregator node hierarchy [130]. The aggregator node collects gradient updates from the working nodes and returns the updated weights, which can lead to a communication burden. At this point, the focus shifts to decentralization and hardware–algorithm co-design. Li et al. [131] implemented the INCEPTIONN framework, which integrates hardware and algorithms, featuring a lightweight, hardware-friendly lossy compression method for floating-point gradient values and a decentralized training algorithm without aggregation nodes; the 2Direction approach by Tyurin et al. [132] addressed distributed convex optimization using bi-directional compressed communication and a customized error feedback mechanism, particularly effective when both uplink and downlink communication is costly for servers and workers; Cao et al. [133] proposed combining gradient variance reduction techniques with compression algorithms to dynamically adjust the weights of error compensation, thereby reducing the cumulative error caused by long-tailed gradient distributions. This error feedback mechanism can effectively mitigate the impact of compression distortion on model training. For techniques that are compatible with gradient compression through a unified analysis framework [134], combining local update strategies under dynamic communication topologies and explicitly separating the compression error term in the theoretical derivation (while weakening the global noise assumption) can guide the adaptive adjustment of compression parameters. This approach helps to maintain communication efficiency while reducing the negative impact of distortion on model convergence; they introduced DC2 framework [135] mainly addressed the communication delay caused by the communication model updates after training iterations. Iterative communication between nodes can hinder communication for distributed non-convex optimization. Yi et al. [136] pointed out a combination of compression and communication techniques for distributed primal pair communication, which can facilitate communication between nodes.

He et al. [137] introduced a key innovation: achieving unbiased compression by lowering the cost for each communication, rather than reducing the number of communication instances, even when compressors used by all working nodes are independent. This is because independence allows the compression error to be effectively controlled, while improved analysis of the ADIANA algorithm demonstrates that these lower bounds are tight; Sketch-fusion SGD [138] utilized the Count-Sketch [139] data structure to augment the scalability and accelerate the training process of distributed deep learning frameworks. This method improves performance by compressing gradients into Count-Sketch structures on a local server, then merging these compressed gradient sketches [128] in an aggregation phase, and finally recovering the critical elements of the gradient for model updating.

Tang et al. [140] presented two compression decentralized training frameworks, one is extrapolation compression (ECD-PSGD), which compresses the extrapolated



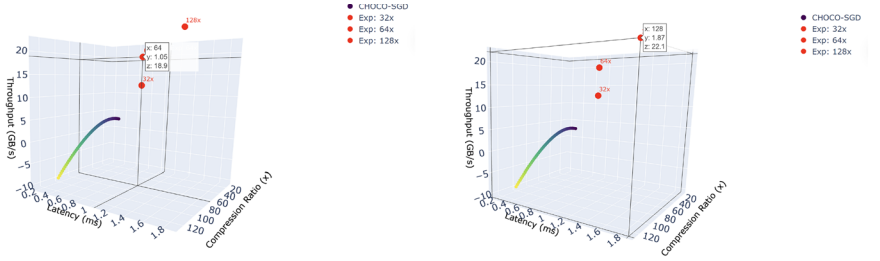
(a) Relationship between Latency and Throughput for CHOCO-SGD Algorithm across Various Compression Ratios

(b) Relationship between Latency and Throughput for CHOCO-SGD Algorithm at 32x Compression Ratio

Fig. 5 Comparative analysis of latency and throughput in CHOCO-SGD algorithm across different compression ratios

values between the last two local models. The other is difference compression (DCD-PSGD), which compresses the differences of local models between successive iterations. The impact on scalability is shown in Table 12; the CHOCO-SGD algorithm [141] is mainly a stochastic gradient descent method [142] for decentralized training using communication compression, and its specific role is to achieve linear acceleration at arbitrarily high compression ratios. And this 3D visualization like highlights key performance metrics of the CHOCO-SGD algorithm. Latency increases logarithmically with compression ratio (0.82 ms at 32× to 1.87 ms at 128×), showing the trade-off between communication efficiency and computational overhead. From Figs. 5 and 6, throughput peaks at 18.9 GB/s under 64× compression but declines to 22.1 GB/s at 128× due to computational saturation. The algorithm's theoretical limits are indicated by a purple performance boundary curve, with an inflection at 75× compression marking the transition between communication and computational bottlenecks. For optimal performance, the 50--80× compression range is recommended to maintain sub-1.2 ms latency and throughput above 18 GB/s, avoiding degradation beyond $\lambda = 100\times$.

The ScaleCom framework [40] (scalable sparsified gradient compression) is analyzed theoretically and demonstrated to provide favorable convergence. It is compatible with full gradient normalization techniques; the study optimizes at the system level for efficient scaling, constructing [117] a scalable system (BytePS-Compress), which blended gradient compression utilizing a new adaptive gradient technique. It also adopts a parameter server architecture and considers bi-directional compression. This high degree of parallelization of compression and decompression improves the processing efficiency of the system by pipelining the operations and utilizing multi-threading; the DeepSpeed framework [143] integrates gradient compression with a sharded optimizer, employing zero redundancy (ZeRO) memory management to avoid information loss due to memory compression or extreme sharding in training models with hundreds of billions of parameters. This approach maintains the integrity of parameter computation and supports the convergence of large models like Turing-NLG. Compared to



(a) Relationship between Latency and Throughput for CHOCO-SGD Algorithm at 64x Compression Ratio

(b) Relationship between Latency and Throughput for CHOCO-SGD Algorithm at 128x Compression Ratio

Fig. 6 Impact of high compression ratios on CHOCO-SGD algorithm performance

traditional methods, DeepSpeed achieves a 3–5 times speedup without any loss of precision.

Minimizing the communication burden associated with gradient and parameter exchanges among nodes is a critical issue in large-scale distributed training. To address this, researchers have investigated a wide range of innovative compression techniques and strategies aimed at speeding up the training process. These approaches encompass communication delay techniques, gradient sparsification methods, binarization schemes, and optimized coding for weight updates, all of which are fine-tuned at the system level to ensure efficient scaling [144]. For instance, the SBC framework proposed by Sattler et al. [129] and the deep gradient compression (DGC) technique developed by Lin et al. [37] demonstrate how to effectively preserve the accuracy of the compression process while significantly reducing communication needs in resource-limited environments. Moreover, the INCEPTIONN framework enables decentralized training and achieves efficient compression of floating-point gradient values through a co-design of hardware and algorithms. Specific decentralization strategies include extrapolation compression and difference compression methods, as well as the LayerFusion algorithm, which reduces communication frequency by employing multilayer fusion techniques. These methods collectively aim to enhance scalability and efficiency by minimizing the volume of data transmitted. In summary, through advanced communication compression [145] and decentralized training techniques, researchers can substantially reduce communication overhead while maintaining model performance, which is essential for effective data transfer management in large-scale distributed systems.

5.2 Federated learning

In federated learning [146], multiple devices or nodes collaborate to train a global model while avoiding the sharing of sensitive data held by each device. In this learning paradigm, data retention is localized, while model updates are transmitted to a centralized server for consolidation, subsequently facilitating the update of the overarching global model [63]. However, because federated learning involves communication across networks, the data upload process can become

Table 12 Summary of SGD compression methods

Method	Key concept	Keywords
ECD-PSGD	Compressing differences in local models	Local model compression, successive iterations
DCD-PSGD	Compressing extrapolated values	Extrapolated values, local models
CHOCO-SGD	Linear acceleration at high compression ratios	Linear acceleration, high compression ratios
CDP-SGD	Communication compression with differential privacy	Communication compression, differential privacy, SGD
Sketch-fusion SGD	Enhancing scalability and training speed	Scalability, training speed, Count-Sketch, distributed deep learning
LIEC-SGD	Local gradient compression with error compensation	Stochastic gradient compression, local error compensation, worker node

a bottleneck, especially in bandwidth-constrained and latency-sensitive application scenarios. Therefore, communication compression techniques are crucial for reducing the latency and bandwidth consumption of federated learning systems.

We always face the following two dilemmas under federated learning:

1. Communication and computational burden: FL places a heavy communication and computing load on participating devices since it requires ongoing local training in addition to periodic global synchronization, especially for battery-constrained mobile devices.
2. Heterogeneous environment adaptation: A flexible compression control method is required to adjust to these heterogeneous settings while maintaining energy efficiency and model accuracy because the processing and communication capacities of the various participating devices vary.

Reducing communication overhead is a key issue in federated learning. Many experiments use more efficient communication protocols and routing algorithms by optimizing network topology. This approach requires costly and difficult modifications to the existing network. In contrast, communication compression directly reduces the amount of data transmitted without altering the network structure, offering higher operability.

The use of dedicated hardware to accelerate the model training and inference process can also reduce the communication requirements, but hardware-accelerated devices are costly and require specific programming and optimization skills, limiting the scope of application. Communication compression can be implemented on general-purpose hardware without additional hardware investment and is compatible with existing hardware acceleration techniques, further improve efficiency.

Distributed computing frameworks optimize task allocation and data transfer, but they require integration and configuration of existing systems, increasing system complexity. Communication compression can be implemented on existing distributed computing frameworks, complementing the optimization strategies of these frameworks and improving overall performance. Opting for communication compression techniques to mitigate the communication overhead in federated learning enhances efficiency, broadens the scope of applicability, and augments the flexibility of the system.

The most notable challenge in federated learning is its high communication costs and the imbalance between efficiency and privacy. The primary solutions involve performing gradient compression and optimization. Li et al. [147] suggest that the core of SoteriaFL integrates generic compression operators and local differential privacy (LDP) to form a versatile private FL framework. The key technique is employing shift compression to optimize communication without compromising accuracy; remote learners and parameter servers (PSs) present a key challenge for the current communication constraints. Model updates must be compressed in order to reduce accuracy loss brought on by communication limitations. Liu et al. [148] efficiently compressed the gradient using a

Table 13 Keywords for communication compression methods in federated learning

Method	Keywords	Reference
SoteriaFL	Generic compression, Local differential privacy (LDP), Private FL framework	[147]
M-amplitude-weighted L2 distortion	Rate distortion, Gradient compression, Communication overhead reduction	[148]
Adaptive control algorithm	Adaptive control, Joint optimization, Error runtime convergence, FedAvg	[149]
Decentralized training and peer-to-peer learning	Decentralized training, Peer-to-peer learning, Bandwidth optimization, Data privacy	[150]
Geometric median-based selection	Geometric median, Filter selection, Efficient communication	[151]
Model sparsification and perturbed compression	Model sparsification, Weight sharing, Perturbed compression, Privacy protection	[152]

rate distortion-inspired method to reduce the communication overhead of model updates while maintaining the accuracy of model training, which is referred to as M-amplitude-weighted L2 distortion, as shown in Table 13.

In terms of client selection and communication period adjustment, Wu et al. [149] highlighted a cooperative optimization strategy to reduce the FedAvg algorithm's error-runtime convergence. To hasten the model's convergence, they created an adaptive control method that can dynamically choose the number of clients and communication duration during FL training; the devices share updates over the network because the FL system is distributed, which could impede communication. The main concerns of Shahid et al. [150] are bandwidth and data privacy, and they explore decentralized training and peer-to-peer learning; parameter interactions during federated learning (FL) tend to lead to frequent parameter communication. This has an additional impact on communication and learning efficiency when coupled with the constrained bandwidth of IoT and edge devices. This methodology enhanced communication efficiency and simultaneously optimizes the performance of the shared global model by prioritizing parameter selection based on their maximal contribution to the global model and their local significance. The geometric median of each layer serves as a benchmark for selecting salient filters within the local model, facilitating subsequent parameter interaction with other nodes to optimize communication efficiency [151]; the team [153] integrated relevant theoretical knowledge for federated learning communication, such as the main communication environments, and proposes resource allocation and common methods.

To further reduce model size and protect model privacy without significantly sacrificing accuracy, Zhu et al. [152] have pointed out an FL framework that integrates model sparsification, weight sharing, and perturbed model compression. In the field of federated learning, where communication compression techniques have become a prominent research topic, the goal is to ensure effective model training and accuracy while protecting data privacy and reducing communication costs. Future research could concentrate on creating more efficient compression algorithms, improving privacy-preserving strategies, and optimizing system designs to improve the performance and scalability of federated learning systems.

In the federated learning (FL) paradigm, a multitude of devices or nodes engage in the collaborative training of a global model, safeguarding the privacy of sensitive data by transmitting model updates in lieu of raw data to a central server for aggregation. However, federated learning involves communication across networks, resulting in a data upload process that can be a performance bottleneck, especially in bandwidth-constrained and latency-sensitive scenarios. Therefore, communication compression techniques are essential in federated learning to effectively reduce latency and bandwidth usage.

5.3 Collective communication systems

Unlike the introduction to communication compression framework design in Sect. 5.1, our focus here shifts to collective communication. We compare the efficiency of traditional collective communication methods with communication compression, emphasizing that communication compression is a superior approach for improving training efficiency.

By employing advanced communication compression techniques, these systems can maintain or improve training quality while reducing bandwidth and time consumption [154]. Below are several major communication compression methods and frameworks, each optimizing data transmission in distributed training through different strategies.

Large message sizes in MPI collective communication are of particular concern, as they can significantly degrade overall parallel performance. Existing research [155] has only applied off-the-shelf fixed-rate loss compressors to MPI collective communication, resulting in poor performance, limited generality, and uncontrolled errors. The C-Coll method [156] utilized bounded error loss compression to significantly reduce message sizes, thereby drastically reducing the communication cost; a customized ultra-fast bounded error loss compressor, SZx [157], is developed to satisfy the specific needs of collective communication; the characteristics are shown in Table 14.

To improve the efficiency of sparse data processing and speed up the training of large-scale deep learning models, OmniReduce [158] combined a block-based gradient compression technique with an efficient streaming aggregation system. By randomly picking the most significant blocks based on the blocks' gradient paradigm,

Table 14 Keywords for communication compression methods of collective communication

Method	Keywords	Reference
C-Coll	Bounded error loss compression, Message size reduction, Communication cost reduction	[156]
SZx	Ultra-fast bounded error loss compression, Collective communication	[157]
OmniReduce	Block-based gradient compression, Streaming aggregation, Data transmission reduction	[158]
1-bit Efficient communication via Adam	1-bit compression, Momentum compression, Communication reduction	[159]
gzccl Framework	GPU clusters, Compression optimization, Collective communication acceleration	[160]
GPU-LCC Framework	GPU clusters, Compression-accelerated communication, Data accuracy	[161]

this method minimizes the quantity of data supplied while preserving training quality.

Tang et al. [159] described a large-scale training algorithm called 1-bit Efficient Communication via Adam, which first uses uncompressed Adam for warm-up at the initial stage. It switches to the compression stage after the variance stabilizes, and achieves up to a 5-fold reduction in communication by transmitting only 1-bit of compressed momentum during the compression stage. This approach also significantly improves the communication efficiency of training, especially in environments with limited network bandwidth.

The gzccl framework which aims to accelerate collective communication on GPU clusters through compression to address the major bottleneck caused by the rapid increase in GPU computing power in modern computing platforms scenario provided some efficient and flexible solutions [160], through compression-accelerated and algorithmic optimization, to solve some of the details of the communication while significantly improving the performance of the collective communication; to address underutilized GPU devices and uncontrollable data distortions, the GPU-LCC framework [161]: depicts a general approach for compression-accelerated collective communication in GPU clusters. This framework seeks to achieve high-performance and high-quality communication by lowering the amount of communication data through compression, thereby lowering communication costs while ensuring data accuracy. The framework [155] aimed to reduce the amount of communication data through compression, thereby reducing communication costs while ensuring data accuracy.

These techniques and frameworks show great potential in dealing with the communication challenges of large-scale distributed training [162]. As computing architectures evolve and data volumes grow, future research will likely focus on further enhancing the efficiency and flexibility of communication compression techniques to better

support the demands of distributed machine learning. This involves developing new compression algorithms, improving error control strategies [163], and optimizing the co-design of hardware and algorithms.

As computing architectures evolve and data volumes grow, the development of more efficient and flexible aggregated communication techniques will be an important direction for future research. These techniques and frameworks show great potential in addressing the communication challenges of large-scale distributed training.

5.4 IoT systems

With the exponential growth in data volume in today's mobile systems, new techniques are needed to increase transmission speed, resilience, and security. In the face of the massive presence of IoT devices and multimodal, high-dimensional data, traditional communication paradigms are no longer sufficient to address these challenges [164]. New communication paradigms are needed that move from the pursuit of accuracy or fidelity to semantic extraction and goal completion. Traditional communication systems focus mainly on the technical aspects and ignore the semantics and validity of the information. Goal-oriented communication improves resource utilization efficiency by considering the end use of the transmitted information.

Target-oriented communication methods particularly applied in Internet of Things (IoT) data compression; this thesis focuses on the application of quantization, which is a fundamental component of data compression systems [165]. In traditional methods, quantization requires adaptation to quantize the input signal. However, for general objective functions, quantization rules for general objective functions must consider the regularity and smoothness of the objective and decision functions; there are also intent-based semantic communication methods, which combine machine learning and reasoning tools by introducing neuro-symbolic artificial intelligence (NeSy AI) to improve the communication efficiency and to reduce the unnecessary data transmission [166].

However, existing compression methods have limitations. Many existing compression methods are only applicable to specific types of data and cannot be broadly applied to data generated by all IoT devices [167], and they cannot ensure the accuracy of recovered data while maintaining the data compression ratio.

5.5 Scientific computing

Communication compression is a critical technique in scientific computing, especially in high-performance computing (HPC) environments where large-scale simulations and data-intensive applications generate vast amounts of data. The quantity of data that must be sent between computing nodes can be greatly decreased with effective communication compression, improving system performance overall, cutting latency, and conserving bandwidth.

Li et al. [123] devised SPERR (SPEck with ERRor bounding), a lossy scientific data compression method for high-performance computing (HPC) environments that provides maximum point-wise error (PWE) tolerance. Based on the SPECK algorithm [169], which uses wavelet transform coefficients for encoding, SPERR

progressively narrows down the range of significant coefficients by partitioning the data into subsets and performing coefficient significance tests. This method efficiently encodes significant coefficients, ensuring PWE-guaranteed data compression suitable for scientific applications that require high accuracy and efficient storage. The study [124] introduced HLRcompress, a high-performance spatial data compression algorithm combining hierarchical low-rank (HLR) approximation and binary compression. By partitioning the 2D data matrix into hierarchical chunks and performing low-rank approximation for each chunk, neighboring low-rank matrices are merged and further compressed. The merged matrix is retained if its low-rank representation requires less memory than the total memory of the sub-blocks; otherwise, the original sub-blocks are preserved. The ZFP algorithm is used for binary compression to ensure optimal memory representation with user-defined precision. HLRcompress efficiently compresses scientific simulation data, significantly improving I/O performance without losing data accuracy.

Fast high-ratio error-limiting lossy compressor FZ-GPU [122] is intended for use in scientific computing applications using GPUs. The approach focuses on achieving high compression ratios and high throughput by exploiting a compression flow including fully parallel quantization, bit shuffling, and a newly designed fast lossless encoding method; it reduces unnecessary data movement between global and shared memory by fusing bit shuffling and encoding operations, and avoids data conflicts in bit operations through warp-level optimization and maximizing the utilization of shared memory. The thesis [168] combined several compression techniques, such as wavelet transforms and dictionary-based methods, to take advantage of the strengths of each technique, as can be seen in Table 15. The hybrid compression approach aims to optimize the compression ratio and computational efficiency.

CFD simulations require large amounts of flow field data to transfer between compute nodes, compression reduces communication overheads and allows more complex simulations to finish in a reasonable amount of time [170]; climate simulations [171] generate large amounts of data. By lowering the quantity of data sent between compute nodes, communication compression can improve the efficiency of simulations and expedite analysis and visualization. Communication compression ensures compression quality while delivering performance gains in scientific computing applications that require high accuracy and efficient storage.

5.6 Multi-model collaboration systems

When multiple AI models collaborate on different devices or locations to solve problems (e.g., multi-robot collaboration, partitioning of information resources, etc.), the communication between models may require compression to enhance collaboration efficiency. With the evolution of big data technology, different types of compression for AI are becoming increasingly necessary. Chen et al. [172] implemented data compression techniques for managing petabytes of data, particularly to improve I/O access speed; Liu et al. [173] developed a multi-model pruning and distillation approach for multilayer convolutional neural networks, which are widely used nowadays. On heterogeneous devices, network pruning, vector quantization, distillation,

Table 15 Communication compression methods in scientific computing

Method	Keywords	Reference
SPERR (SPEck with ERRor bounding)	Error bounded, waveform transformation, SPECK coding	[123]
HLRcompress	Hierarchical low-rank approximation, binary compression	[124]
FZ-GPU	Fast, high ratio, GPU, error bounded	[122]
Enhanced prediction algorithms	Spatial correlation, temporal correlation, accuracy	[168]
Error-bounded compression	Maximum error threshold, controlled precision	[168]
Adaptive quantization	Local variance, dynamic quantization, high compression	[168]
Hybrid compression techniques	Wavelet transforms, dictionary-based methods, optimization	[168]
Parallelization and scalability optimization	Multi-threading, parallel execution, modern architectures	[168]
Adaptability to diverse data types	Structured grids, unstructured meshes, point clouds	[168]

and other compression techniques [174] are primarily applied to accelerate multi-model reasoning even under resource constraints.

Multi-model architectures have smaller memory and computational resource requirements due to their flexibility and lightweight nature compared to a single large model. Designing relevant parallel architectures can enhance their performance and save memory and computational resources.

5.7 Model deployment

Model compression is required when distributing trained models to edge devices (such as mobile phones, IoT devices, etc.) because of these devices' constrained storage, processing capacity, and network bandwidth.

For distributed deep learning inference for the Internet of Things (IoT), Bhardwaj et al. [175] investigated a memory and communication-aware model compression technique known as network of networks (NoNN). It enables the compression of huge teacher models into several independent, highly compressed student networks that each learn certain aspects of the teacher function, resulting in a highly parallel architecture that enhances accuracy while reducing communication costs. For deploying these model compression and communication compression approaches [176] on edge devices, it is essential to maintain a high level of accuracy while significantly reducing model memory and computation requirements, addressing the challenge of deploying complex deep learning models in resource-constrained environments.

5.8 Model update and maintenance

In the case of online learning or continuous deployment of models, model parameters need periodic updates, where compression of model parameters can reduce the communication load required for updates [177]. As the number of artificial intelligence (AI) applications increases, more and more DNNs must be run on edge devices. Following deployment, the DNN model on the edge device must be updated for a variety of practical reasons (for example, model refinement, conceptual drift, or major changes in the learning goal). Chen et al. [178] developed an update compression approach for DNNs on edge devices. The main concept is to optimize the few extra parameters needed to recreate the model on edge devices while maintaining the current model's existing knowledge. By using only half the update size of previous approaches, this method usually achieves the same accuracy when compared to similar techniques employed in federated learning.

6 Libraries

Modern distributed deep learning relies heavily on optimized communication libraries to efficiently coordinate computation across multiple devices. This section examines three pivotal technologies that address these challenges: NCCL, a GPU-optimized collective communication library; Gloo, a flexible CPU/GPU communication framework; and Horovod, a distributed training framework leveraging efficient all-reduce algorithms. Each solution incorporates unique optimization strategies while supporting communication compression techniques to mitigate bandwidth constraints in large-scale deployments. The detailed exposition for each technology will begin with the key information from Fig. 7. These libraries collectively form the backbone of contemporary distributed training systems, enabling scalable and efficient parallel computation across diverse hardware configurations.

6.1 NCCL

NCCL (NVIDIA Collective Communications Library) is a high-performance communications library developed by NVIDIA to accelerate and optimize the process of exchanging data in a multi-GPU environment [179]. Designed for deep learning and high-performance computing (HPC) applications, the library enables efficient data synchronization across multiple GPUs [180], supporting a variety of multi-GPU configurations including NVIDIA GPU clusters. NCCL provides a range of CUDA-based APIs that enable developers to easily parallelize data processing and communication across different GPU devices, thus accelerating the overall computing process.

NCCL incorporates optimization strategies like DMA for minimal CPU overhead, and smart network topology adaptation to enhance communication efficiency. Although it prioritizes bandwidth utilization and latency reduction, it pairs synergistically with communication compression techniques to manage data volume in

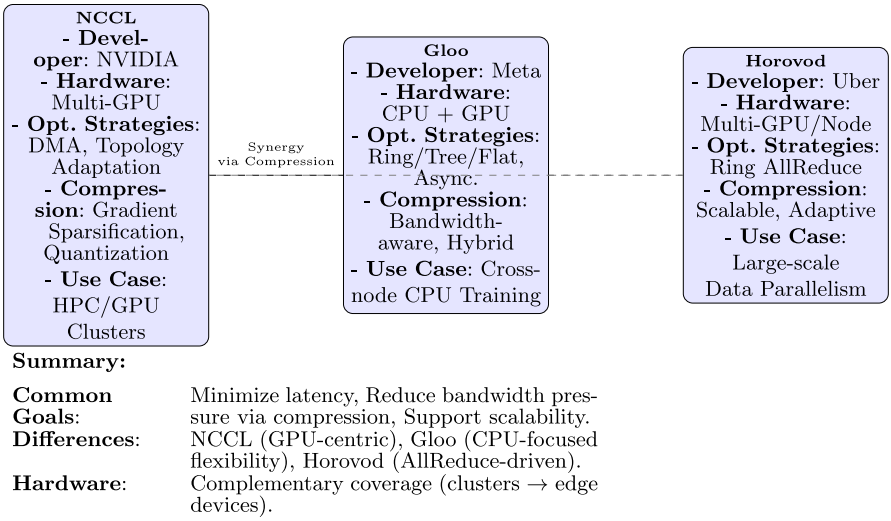


Fig. 7 Summary of NCCL, Gloo, and Horovod: key features and optimization strategies for distributed deep learning

bandwidth-constrained scenarios, such as gradient sparsification and quantization. This hybrid approach not only boosts performance but also simplifies parallel programming and offers economic benefits by reducing the need for expensive hardware upgrades and lowering energy use. Overall, NCCL, supplemented by communication compression, is a formidable solution for efficient multi-GPU computation, propelling advancements in high-performance computing tasks.

6.2 Gloo

Gloo [181] is an open-source aggregate communication library designed for distributed machine learning and deep learning tasks to efficiently synchronize data across multiple compute nodes. Developed and sustained by Facebook's Artificial Intelligence Research Group (FAIR), as referenced in [182] and [183], Gloo is designed to optimize and accelerate communication efficiency during large-scale distributed training [184]. Unique to other aggregate communication libraries such as NVIDIA's NCCL, Gloo is optimized for CPU environments but also supports GPU environments, providing a flexible and efficient way to handle cross-device data communication.

The library's optimization strategies encompass intelligent selection of communication modes like Ring, Tree, and Flat to suit specific network topologies, enhancing memory efficiency and supporting asynchronous communication for simultaneous computation and data transfer. While Gloo is adept at handling various hardware and is optimized for communication, it also addresses potential multi-node performance bottlenecks by integrating communication compression techniques, which reduce data volume and ease network bandwidth constraints.

Gloo's commitment to the open-source community and its continuous evolution alongside technological advancements solidify its position as a pivotal asset in the realm of distributed computing. It is anticipated to further expand its utility and optimization in high-performance computing applications, contributing significantly to the progress of the industry.

6.3 Horovod

Horovod [185], an open-source framework for distributed deep learning training, was developed by Uber's technical team and released to the public domain in 2017, as documented in [186]. It works based on data parallelism and aims to make deep learning model training in multi-GPU and multi-node environments simpler and more efficient. Horovod's design philosophy is to increase the speed of model training on multi-GPUs and multi-nodes through a set of highly efficient distributed training strategies while minimizing modifications to the original model code.

Designed with an efficient communication strategy leveraging the Ring All-Reduce algorithm, Horovod ensures high performance in data synchronization with low latency, making it particularly adept at handling large-scale datasets and complex models. Its fine-grained GPU resource management and task scheduling further optimize computational resource utilization, significantly reducing model training times and enabling faster iteration and improvement in models in applications like image recognition and natural language processing.

Despite challenges including network bandwidth and latency in large-scale clusters, Horovod addresses these through communication compression, enhancing scalability and mitigating performance bottlenecks. As it continues to advance with community contributions, Horovod remains a vital tool for the acceleration of deep learning research and applications, supporting the industry's growing computational demands and driving innovation forward.

7 Future directions and conclusion

7.1 Future directions

Communication compression reduces communication overheads while incurring additional computational overheads due to methods such as quantization, sparsification, and compressive coding, as well as degradation of accuracy due to the inevitable loss of detailed information during compression. Although we are analyzing these two bottlenecks separately here, they often arise simultaneously, as highlighted in several of the mentioned papers. For example, in a distributed training environment, nodes need to frequently exchange gradient information. If we focus only on reducing communication overhead while ignoring the effects of increased computational overhead and information loss, it may lead to a degradation of overall system performance.

Adaptive compression for advanced interconnects New interconnect technologies such as NVLink, CXL (Compute Express Link) and UALink (Unified Accelerator Link) are rapidly evolving in modern HPC systems. These interconnects provide higher bandwidth and lower latency, making it possible to investigate new methods of communication compression to improve overall system performance further. NVLink is a high-bandwidth, low-latency interconnect primarily used to connect NVIDIA GPUs to other processing units. New communication compression methods based on it can take full advantage of its high bandwidth. For instance, the amount of data transported across GPUs can be decreased, increasing the effectiveness of distributed training, by compressing intermediate data and model parameters. In addition, real-time compression and decompression are made possible by NVLink's reduced latency, creating further opportunities for future data transfer performance optimization.

Bandwidth-optimized compression in AI training clusters In high-density AI training clusters, which integrate multiple GPUs or AI accelerators, high-bandwidth interconnects, and require high throughput, researching communication compression methods oriented toward high bandwidth and high throughput is crucial.

1. For the communication characteristics of high-density AI training clusters, adaptive compression algorithms are an important research direction. These algorithms can dynamically adjust the compression ratio and method based on current network bandwidth, data transfer volume, and computational resources. For instance, when network bandwidth is abundant, one can opt for a low compression ratio or lossless compression, whereas when bandwidth is limited, one can utilize a high compression ratio or implement lossy compression techniques to diminish the volume of data transmitted.
2. Utilizing multi-level compression strategies allows for data compression at different levels. For example, in communications between accelerators, lightweight and fast compression algorithms can be used, while in communications between servers, more efficient deep compression algorithms can be applied. This tiered compression strategy can maximize the utilization of the system's computational resources and network bandwidth, reducing communication overhead.
3. Against the backdrop of high throughput demands, parallel and pipeline compression methods can significantly improve data transfer efficiency. By parallelizing compression and decompression operations, or processing them in a pipeline during data transmission, the total communication time can be reduced, enhancing training speed.
4. Considering the hardware characteristics of high-density AI training clusters, research can be conducted on hardware-accelerated compression methods, for example leveraging the parallel computing capabilities of GPUs to achieve rapid data compression, or using dedicated compression hardware accelerators (such as hardware compression modules in TPUs) to reduce the computational overhead of compression and decompression.

To verify these methods, experiments can be conducted on actual high-density AI training clusters. For example, they are deploying the aforementioned compression algorithms and strategies on NVIDIA's DGX systems or Huawei's Ascend training servers, combining the algorithms, and measuring their impact on training speed, communication overhead, and system resource utilization. Through experimental data, these compression methods can be further optimized and improved to adapt to different training tasks and system configurations.

Distributed compression for cloud-edge-device network Within the intricate network ecosystem of cloud-edge devices, the challenge of communication compression in distributed training and inference emerges as a pivotal area of research. Distributed training and inference pertain to the delegation of training or inference operations to a multitude of computational nodes. This approach capitalizes on the robust computational resources of cloud infrastructure, the geographical propinquity of edge devices, and the real-time responsiveness of terminal devices. This environment involves multiple computing nodes, including cloud servers, edge devices, and terminal devices, which collaborate through different network connections. Typically, in scenarios such as autonomous driving, intelligent surveillance, or industrial Internet of Things, communication compression technology can improve their performance and reliability. Looking ahead, it is imperative to concentrate on the development of technologies capable of dynamically adjusting compression strategies in response to network conditions and task-specific demands. Future research should delve into more efficacious methods for compressing deep learning models without compromising their performance. Additionally, there is a need to investigate unified compression methodologies applicable to diverse data types, including images, text, and audio. Addressing the challenges of communication compression in distributed training and inference within the intricate cloud-edge-device network environment is essential for effective solutions.

Quantum computing reconfigures communications compression Quantum computing is reshaping communication compression via its unique properties. Quantum superposition allows a single qubit to encode exponential-scale information (e.g., n qubits representing 2^n states, based on quantum information density theory [187], and quantum autoencoders achieving a compression ratio of 90% [188]), while quantum entanglement realizes "superdense coding" through nonlocal correlations [189], providing a new paradigm for lossless compression. Meanwhile, quantum parallelism accelerates traditional compression algorithms; for example, federated quantum compression frameworks reduce pattern-matching complexity to $O(\log N)$ [190]. In the dimension of efficiency improvement, quantum compression significantly reduces resource consumption: pre-transmission quantum state compression can reduce the quantum channel load by over 50% [191], while hybrid protocols enhance the efficiency of classical-to-quantum data conversion [192]. In terms of security, quantum error-correcting codes [193] suppress noise through redundant quantum state encoding, increasing communication fault tolerance by 3–5 orders of magnitude.

In application scenarios, the quantum internet relies on compression technology to reduce teleportation bandwidth requirements [194]. Distributed quantum computing leverages compression of intermediate states (e.g., results of quantum gate operations) to reduce cross-node communication by 80% [195, 196]. Quantum sensor networks use real-time compression algorithms to process petabyte-scale quantum data streams, with real-time quantum compression algorithms increasing LIGO's data throughput by tenfold [197]. Despite challenges such as the fragility of quantum states limiting lossy compression, cutting-edge research focuses on quantum reinforcement learning-driven dynamic compression frameworks [198] and co-design of photonic–superconducting quantum hardware for compression [199]. In the future, quantum compression will break through the entropy limits of classical information theory, propelling communication technologies from the "Shannon era" into a new era of "quantum–classical symbiosis."

Future development trends in communication compression can be analyzed from innovative application scenarios. Effective compression technology can lower the cost of data transmission and storage and boost data processing efficiency when combined with machine learning training and communication. In high real-time requirements scenarios such as the Internet of Things, unmanned driving, and intelligent manufacturing, real-time data processing is essential. It is crucial to optimize communication and computation overhead to ensure that compression technology meets the needs of real-time processing.

For optimizing computational overhead, the design of lightweight compression methods, combined with hardware accelerators, can reduce computational energy consumption. To improve accuracy, research and development should focus on retaining important features of the data under high compression ratios, ensuring that the compressed data still maintains high accuracy after decompression. Additionally, introducing an error compensation mechanism during the compression process can correct errors generated during compression, thereby improving accuracy.

7.2 Conclusion

In the field of distributed machine learning and optimization, communication compression algorithms play a pivotal role in minimizing communication bottlenecks by decreasing the amount of data that needs to be shared among nodes. But because of the extra compression and decompression steps, there are a computational overhead and a chance of errors because of the compression procedures. The development of more efficient compression algorithms is emerging as a promising trend, aiming to achieve more effective communication compression in the future while maintaining model performance through approaches such as optimization algorithms, hardware acceleration, hybrid strategies, and accuracy recovery.

Author contributions The paper properly credits the meaningful contributions of all authors. All authors have been personally and actively involved in substantial work leading to the paper and will take public responsibility for its content.

Funding The research was partially funded by The National Key Research and Development Program of China (2023YFA1011704).

Data availability No datasets were generated or analyzed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Li M, Andersen DG, Smola A, Yu K (2014) Communication efficient distributed machine learning with the parameter server. Advances in Neural Information Processing Systems 27
2. Zinkevich M, Weimer M, Li L, Smola A (2010) Parallelized stochastic gradient descent. Advances in Neural Information Processing Systems 23
3. Lian X, Zhang C, Zhang H, Hsieh C-J, Zhang W, Liu, J (2017) Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. Advances in Neural Information Processing Systems **30**
4. Seide F, Fu H, Droppo J, Li G, Yu D (2014) 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In: Fifteenth Annual Conference of the International Speech Communication Association
5. Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, Almeida D, Altenschmidt J, Altman S, Anadkat S et al (2023) Gpt-4 technical report. arXiv preprint arXiv:2303.08774
6. Ben-Nun T, Hoefler T (2019) Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. ACM Comput Surv (CSUR) 52(4):1–43
7. Bertsekas DP, Tsitsiklis JN (2003) Parallel and distributed computation: numerical methods (2nd ed.). Athena Scientific
8. Wright J, Ma Y, Mairal J, Sapiro G, Huang TS, Yan S (2010) Sparse representation for computer vision and pattern recognition. Proc IEEE 98(6):1031–1044
9. Yaguchi A, Suzuki T, Nitta S, Sakata Y, Tanizawa A (2019) Scalable deep neural networks via low-rank matrix factorization. arXiv preprint arXiv:1910.13141
10. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2016) Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710
11. Wangni J, Wang J, Liu J, Zhang T (2018) Gradient sparsification for communication-efficient distributed optimization. Advances in Neural Information Processing Systems 31
12. Li S, Hoefler T (2022) Near-optimal sparse allreduce for distributed deep learning. In: Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp 135–149
13. Abrahamyan L, Chen Y, Bekoulis G, Deligiannis N (2021) Learned gradient compression for distributed deep learning. IEEE Trans Neural Netw Learn Syst 33(12):7330–7344
14. Huang X, Chen Y, Yin W, Yuan K (2022) Lower bounds and nearly optimal algorithms in distributed learning with communication compression. Adv Neural Inf Process Syst 35:18955–18969
15. Luzuriaga JE, Perez M, Boronat P, Cano JC, Calafate C, Manzoni P (2015) A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In: 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC). IEEE, pp 931–936
16. Winters J (1987) On the capacity of radio communication systems with diversity in a Rayleigh fading environment. IEEE J Sel Areas Commun 5(5):871–878
17. Jiang J, Hu L (2020) Decentralised federated learning with adaptive partial gradient aggregation. CAAI Trans Intell Technol 5(3):230–236
18. Deng Y, Kamani MM, Mahdavi M (2020) Adaptive personalized federated learning. arXiv preprint arXiv:2003.13461
19. Jiang Y, Zhu Y, Lan C, Yi B, Cui Y, Guo C (2020) A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pp 463–479

20. Wang Z, Wen M, Xu Y, Zhou Y, Wang JH, Zhang L (2023) Communication compression techniques in distributed deep learning: a survey. J Syst Architect 142:102927
21. Hanna OA, Ezzeldin YH, Fragouli C, Diggavi S (2020) Quantizing data for distributed learning. arXiv preprint arXiv:2012.07913
22. Liu H-S, Chuang C-C, Lin C-C, Chang R-I, Wang C-H, Hsieh C-C (2011) Data compression for energy efficient communication on ubiquitous sensor network. Tamkang J Sci Eng 14(3):345–354
23. Li H, Xu Y, Chen J, Dwivedula R, Wu W, He K, Akella A, Kim D (2024) Accelerating distributed deep learning using lossless homomorphic compression. arXiv preprint arXiv:2402.07529
24. Makarenko M, Gasanov E, Islamov R, Sadiev A, Richtárik P (2022) Adaptive compression for communication-efficient distributed training. arXiv preprint arXiv:2211.00188
25. Wang Z, Wu XC, Xu Z, Ng TE (2023) Cupcake: a compression optimizer for scalable communication-efficient distributed training. In: Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys' 23). Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys' 23)
26. Chen C-C, Chou Y-M, Chou J (2023) Phy: A performance-driven hybrid communication compression method for distributed training. J Parallel Distrib Comput 180:104719
27. Yan G, Li T, Huang S-L, Lan T, Song L (2022) Ac-sgd: Adaptively compressed sgd for communication-efficient distributed learning. IEEE J Sel Areas Commun 40(9):2678–2693
28. Stich SU (2018) Local sgd converges fast and communicates little. arXiv preprint arXiv:1805.09767
29. Tsuzuku Y, Imachi H, Akiba T (2018) Variance-based gradient compression for efficient distributed deep learning. arXiv preprint arXiv:1802.06058
30. Agarwal S, Wang H, Venkataraman S, Papailiopoulos D (2022) On the utility of gradient compression in distributed training systems. Proc Mach Learn Syst 4:652–672
31. Khirirat S, Feyzmahdavian HR, Johansson M (2018) Distributed learning with compressed gradients. arXiv preprint arXiv:1806.06573
32. Karimireddy SP, Rebjock Q, Stich S, Jaggi M (2019) Error feedback fixes signsgd and other gradient compression schemes. In: International Conference on Machine Learning. PMLR, pp 3252–3261
33. Xu H, Ho C-Y, Abdelmoniem AM, Dutta A, Bergou EH, Karatsenidis K, Canini M, Kalnis P (2020) Compressed communication for distributed deep learning: survey and quantitative evaluation. Technical report
34. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 770–778
35. Konecnỳ J, McMahan HB, Yu FX, Richtárik P, Suresh AT, Bacon D (2016) Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492
36. Lim H, Andersen DG, Kaminsky M (2019) 3lc: Lightweight and effective traffic compression for distributed machine learning. Proc Mach Learn Syst 1:53–64
37. Lin Y, Han S, Mao H, Wang Y, Dally W (2017) Deep gradient compression: reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887
38. Ren Y, Cao Y, Ye C, Cheng X (2023) Two-layer accumulated quantized compression for communication-efficient federated learning: Tlaqc. Sci Rep 13(1):11658
39. Tang H, Yu C, Lian X, Zhang T, Liu J (2019) Doublesqueeze: parallel stochastic gradient descent with double-pass error-compensated compression. In: International Conference on Machine Learning. PMLR, pp 6155–6165
40. Chen C-Y, Ni J, Lu S, Cui X, Chen P-Y, Sun X, Wang N, Venkataramani S, Srinivasan VV, Zhang W et al (2020) Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training. Adv Neural Inf Process Syst 33:13551–13563
41. Ström N (2015) Scalable distributed dnn training using commodity gpu cloud computing. In Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH 2015) (pp. 1488–1492). ISCA
42. Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, Li HT (2017) Ternary gradients to reduce communication in distributed deep learning. arXiv preprint arXiv:1705.07878
43. Yu R, Li A, Chen C-F, Lai J-H, Morariu VI, Han X, Gao M, Lin C-Y, Davis LS (2018) Nisp: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203
44. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

45. Hoefler T, Alistarh D, Ben-Nun T, Dryden N, Peste A (2021) Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. J Mach Learn Res 22(241):1–124
46. Luo J-H, Wu J (2020) Neural network pruning with residual-connections and limited-data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 1458–1467
47. Weng Z, Liu K, Zheng Z (2023) Cattle face detection method based on channel pruning yolov5 network and mobile deployment. J Intell Fuzzy Syst 45(6):10003–10020
48. Sajjad H, Dalvi F, Durrani N, Nakov P (2023) On the effect of dropping layers of pre-trained transformer models. Comput Speech Lang 77:101429
49. Wang G, Qin H, Jacobs SA, Holmes C, Rajbhandari S, Ruwase O, Yan F, Yang L, He Y (2023) Zero++: extremely efficient collective communication for giant model training. arXiv preprint arXiv:2306.10209
50. Dean J, Corrado G, Monga R, Chen K, Devin M, Mao M, Ranzato M, Senior A, Tucker P, Yang K et al (2012) Large scale distributed deep networks. Advances in Neural Information Processing Systems 25
51. Recht B, Re C, Wright S, Niu F (2011) Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. Advances in Neural Information Processing Systems 24
52. Lian X, Zhang W, Zhang C, Liu J (2018) Asynchronous decentralized parallel stochastic gradient descent. In: International Conference on Machine Learning. PMLR, pp 3043–3052
53. Stich SU, Cordonnier J-B, Jaggi M (2018) Sparsified sgd with memory. Advances in Neural Information Processing Systems 31
54. Wu J, Huang W, Huang J, Zhang T (2018) Error compensated quantized sgd and its applications to large-scale distributed optimization. In: International Conference on Machine Learning. PMLR, pp 5325–5333
55. Alistarh D, Grubic D, Li J, Tomioka R, Vojnovic M (2017) QSGD: communication-efficient sgd via gradient quantization and encoding. Advances in Neural Information Processing Systems 30
56. Ramezani-Kebrya A, Faghri F, Markov I, Aksenov V, Alistarh D, Roy DM (2021) Nuqsgd: Provably communication-efficient data-parallel sgd via nonuniform quantization. J Mach Learn Res 22(114):1–43
57. Moritz P, Nishihara R, Stoica I, Jordan MI (2015) Sparknet: training deep networks in spark. arXiv preprint arXiv:1511.06051
58. Du Y, Yang S, Huang K (2020) High-dimensional stochastic gradient quantization for communication-efficient edge learning. IEEE Trans Signal Process 68:2128–2142
59. Abdi A, Fekri F (2020) Quantized compressive sampling of stochastic gradients for efficient communication in distributed deep learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 34, pp 3105–3112
60. Dryden N, Moon T, Jacobs SA, Van Essen B (2016) Communication quantization for data-parallel training of deep neural networks. In: 2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC). IEEE, pp 1–8
61. Malekijoo A, Fadaeieslam MJ, Malekijou H, Homayounfar M, Alizadeh-Shabdiz F, Rawassizadeh R (2021) Fedzip: a compression framework for communication-efficient federated learning. arXiv preprint arXiv:2102.01593
62. Haddadpour F, Kamani MM, Mokhtari A, Mahdavi M (2021) Federated learning with compression: unified analysis and sharp guarantees. In: International Conference on Artificial Intelligence and Statistics. PMLR, pp 2350–2358
63. Qu L, Song S, Tsui C-Y (2022) Feddq: Communication-efficient federated learning with descending quantization. In: GLOBECOM 2022-2022 IEEE Global Communications Conference. IEEE, pp 281–286
64. Oh Y, Lee N, Jeon Y-S, Poor HV (2022) Communication-efficient federated learning via quantized compressed sensing. IEEE Trans Wirel Commun 22(2):1087–1100
65. Jonkman JA, Sherson T, Heusdens R (2018) Quantisation effects in distributed optimisation. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp 3649–3653
66. Konečnỳ J, Richtárik P (2018) Randomized distributed mean estimation: accuracy vs. communication. Front Appl Math Stat 4:62
67. Berger CR, Wang Z, Huang J, Zhou S (2010) Application of compressive sensing to sparse channel estimation. IEEE Commun Mag 48(11):164–174

68. Mitra A, Richards JA, Bagchi S, Sundaram S (2021) Distributed inference with sparse and quantized communication. IEEE Trans Signal Process 69:3906–3921
69. Eghlidi NF, Jaggi M (2020) Sparse communication for training deep networks. arXiv preprint arXiv:2009.09271
70. Sahu AN, Dutta A, Abdelmoniem AM, Banerjee T, Canini M, Kalnis P (2021) Rethinking gradient sparsification as total error minimization. Adv Neural Inf Process Syst 34:8133–8146
71. Wang H, Sievert S, Liu S, Charles Z, Papailiopoulos D, Wright S (2018) Atomo: communication-efficient learning via atomic sparsification. Advances in Neural Information Processing Systemss 31
72. Shi S, Wang Q, Chu X, Li B, Qin Y, Liu R, Zhao X (2020) Communication-efficient distributed deep learning with merged gradient sparsification on gpus. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, pp 406–415
73. Mishchenko K, Gorbunov E, Takáč M, Richtárik P (2019) Distributed learning with compressed gradient differences. arXiv preprint arXiv:1901.09269
74. Zhen K, Nguyen HD, Chang F-J, Mouchtaris A, Rastrow A (2021) Sparsification via compressed sensing for automatic speech recognition. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp 6009–6013
75. Ozfatura E, Ozfatura K, Gündüz D (2021) Time-correlated sparsification for communication-efficient federated learning. In: 2021 IEEE International Symposium on Information Theory (ISIT). IEEE, pp 461–466
76. Shi S, Zhao K, Wang Q, Tang Z, Chu X (2019) A convergence analysis of distributed SGD with communication-efficient gradient sparsification. In: IJCAI, pp 3411–3417
77. Hou J, Chau L-P, Magnenat-Thalmann N, He Y (2015) Sparse low-rank matrix approximation for data compression. IEEE Trans Circuits Syst Video Technol 27(5):1043–1054
78. Vogels T, Karimireddy SP, Jaggi M (2019) Powersgd: practical low-rank gradient compression for distributed optimization. Advances in Neural Information Processing Systems
79. Bernstein J, Wang Y-X, Azizzadenesheli K, Anandkumar A (2018) SIGNSGD: compressed optimisation for non-convex problems. In: International Conference on Machine Learning. PMLR, pp 560–569
80. Vogels T, Karimireddy SP, Jaggi M (2020) Powergossip: practical low-rank communication compression in decentralized deep learning. arXiv preprint arXiv:2008.01425
81. Qiao Z, Yu X, Zhang J, Letaief KB (2021) Communication-efficient federated learning with dual-side low-rank compression. arXiv preprint arXiv:2104.12416
82. Dong W, Shi G, Li X, Ma Y, Huang F (2014) Compressive sensing via nonlocal low-rank regularization. IEEE Trans Image Process 23(8):3618–3632
83. Chen P, Yu H-F, Dhillon I, Hsieh C-J (2021) Drone: data-aware low-rank compression for large NLP models. Adv Neural Inf Process Syst 34:29321–29334
84. Idelbayev Y, Carreira-Perpinán MA (2020) Low-rank compression of neural nets: Learning the rank of each layer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 8049–8059
85. Jacob M, Mani M, Ye J (2019) Structured low-rank algorithms: theory, MR applications, and links to machine learning, arXiv preprint
86. Young T, Hazarika D, Poria S, Cambria E (2018) Recent trends in deep learning based natural language processing. IEEE Comput Intell Mag 13(3):55–75
87. Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805
88. Xiao X, Shen S, Bao Q, Rong H, Liu K, Wang Z, Liu J (2024) Cora: Optimizing low-rank adaptation with common subspace of large language models. arXiv preprint arXiv:2409.02119
89. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibescu M, Liotta A (2018) Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat Commun 9(1):2383
90. Shen X, Liu W, Tsang I, Shen F, Sun Q-S (2017) Compressed k-means for large-scale clustering. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 31
91. Mao H, Dally W (2016) Deep compression: compressing deep neural. In: ICLR
92. Moffat A (2019) Huffman coding. ACM Comput Surv CSUR) 52(4):1–35
93. Han S, Mao H, Dally W (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149

94. VidyaSagar M, Victor JR (2013) Modified run length encoding scheme for high data compression rate. Int J Adv Res Comput Eng Technol (IJARCET) 2:12
95. Howard PG, Vitter JS (1994) Arithmetic coding for data compression. Proc IEEE 82(6):857–865
96. Ma Z, Zhu H, He Z, Lu Y, Song F (2022) Deep lossless compression algorithm based on arithmetic coding for power data. Sensors 22(14):5331
97. Wang L, Yoon K-J (2021) Knowledge distillation and student–teacher learning for visual intelligence: a review and new outlooks. IEEE Trans Pattern Anal Machine Intell 44(6):3048–3068
98. Zhang Y, Xiang T, Hospedales TM, Lu H (2018) Deep mutual learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 4320–4328
99. Xue H, Ren K (2021) Recent research trends on model compression and knowledge transfer in cnns. In: 2021 IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE). IEEE, pp 136–142
100. An S, Liao Q, Lu Z, Xue J-H (2022) Efficient semantic segmentation via self-attention and self-distillation. IEEE Trans Intell Transp Syst 23(9):15256–15266
101. Ji M, Shin S, Hwang S, Park G, Moon I-C (2021) Refine myself by teaching myself: Feature refinement via self-knowledge distillation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 10664–10673
102. Mou T, Jiang X, Li J, Yan B, Chen Q, Zhang T, Huang W, Gao C, Chen Y (2023) FedRAMP: decentralized federated learning with a feature attention based multi-teacher knowledge distillation for healthcare. In: 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, pp 1246–1253
103. Lee Y, Wu W (2024) QMKD: atwo-stage approach to enhance multi-teacher knowledge distillation. In: 2024 International Joint Conference on Neural Networks (IJCNN). IEEE, pp 1–7
104. Liu H, Wang Y, Liu H, Sun F, Yao A (2024) Small scale data-free knowledge distillation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 6008–6016
105. Zhao K, Zhao M (2024) Self-supervised quantization-aware knowledge distillation. arXiv preprint arXiv:2403.11106
106. Stanton S, Izmailov P, Kirichenko P, Alemi AA, Wilson AG (2021) Does knowledge distillation really work? Adv Neural Inf Process Syst 34:6906–6919
107. Feng Q, Li W, Lin T, Chen X (2024) Align-kd: Distilling cross-modal alignment knowledge for mobile vision-language model. arXiv preprint arXiv:2412.01282
108. Ren J, Rajbhandari S, Aminabadi RY, Ruwase O, Yang S, Zhang M, Li D, He Y (2021) {Zero-offload}: Democratizing {billion-scale} model training. In: 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp 551–564
109. Arora R, Marinov TV, Ullah E (2020) Private stochastic convex optimization: efficient algorithms for non-smooth objectives. arXiv preprint arXiv:2002.09609
110. Pan J, Bulat A, Tan F, Zhu X, Dudziak L, Li H, Tzimiropoulos G, Martinez B (2022) EdgeViTs: competing light-weight cnns on mobile devices with vision transformers. In: European Conference on Computer Vision. Springer, Berlin, pp 294–311
111. Qu R, Bo P, Zhang Y, Wang T, Zhang W, Zhang L (2022) Security verification of 1553b bus components used in aerospace. In: 2022 IEEE 4th International Conference on Power, Intelligent Computing and Systems (ICPICS). IEEE, pp 486–489
112. Hanif MA, Shafique M (2023) Cross-layer optimizations for efficient deep learning inference at the edge. In: Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Software Optimizations and Hardware/Software Codesign. Springer, Cham, pp 225–248
113. Song J, Yim J, Jung J, Jang H, Kim H-J, Kim Y, Lee J (2023) Optimus-cc: efficient large nlp model training with 3d parallelism aware communication compression. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol 2 pp 560–573
114. Islamov R, Qian X, Hanzely S, Safaryan M, Richtárik P (2023) Distributed newton-type methods with communication compression and bernoulli aggregation. Trans Mach Learn Res. https://doi.org/10.48550/arXiv.2206.03588
115. Wang Z, Duan Q, Xu Y, Zhang L (2024) An efficient bandwidth-adaptive gradient compression algorithm for distributed training of deep neural networks. J Syst Architect 150:103116
116. Zhang L, Zhang L, Shi S, Chu X, Li B (2023) Evaluation and optimization of gradient compression for distributed deep learning. In: 2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS). IEEE, pp 361–371

117. Zhong Y, Xie C, Zheng S, Lin H (2021) Compressed communication for distributed training: adaptive methods and system. arXiv preprint arXiv:2105.07829
118. Huang X, Li P, Li X (2023) Stochastic controlled averaging for federated learning with communication compression. arXiv preprint arXiv:2308.08165
119. Zheng S, Huang Z, Kwok J (2019) Communication-efficient distributed blockwise momentum sgd with error-feedback. Advances in Neural Information Processing Systems 32
120. Li L, Shi D, Hou R, Li H, Pan M, Han Z (2021) To talk or to work: flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, pp 1–10
121. Rammal A, Gruntkowska K, Fedin N, Gorbunov E, Richtárik P (2024) Communication compression for byzantine robust learning: new efficient algorithms and improved rates. In: International Conference on Artificial Intelligence and Statistics. PMLR, pp 1207–1215
122. Zhang B, Tian J, Di S, Yu X, Feng Y, Liang X, Tao D, Cappello F (2023) FZ-GPU: a fast and high-ratio lossy compressor for scientific computing applications on gpus. In: Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing, pp 129–142
123. Li S, Lindstrom P, Clyne J (2023) Lossy scientific data compression with sperr. In: 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 1007–1017
124. Kriemann R, Ltaief H, Luong MB, Pérez FEH, Im HG, Keyes D (2022) High-performance spatial data compression for scientific applications. In: European Conference on Parallel Processing. Springer, Berlin, pp 403–418
125. Liu Y, Jia W, Yang T, Yin M, Jin S (2024) Enhancing lossy compression through cross-field information for scientific applications. In: SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 300–308
126. Shen L, Sun Y, Yu Z, Ding L, Tian X, Tao D (2023) On efficient training of large-scale deep learning models: a literature review. arXiv preprint arXiv:2304.03589
127. Dowe DL, Hernández-Orallo J, Das PK (2011) Compression and intelligence: social environments and communication. In: Artificial General Intelligence: 4th International Conference, AGI 2011, Mountain View, CA, USA, August 3-6, 2011. Proceedings 4. Springer, Berlin, pp 204–211
128. Hanzely F, Mishchenko K, Richtárik P (2018) Sega: variance reduction via gradient sketching. Advances in Neural Information Processing Systems 31
129. Sattler F, Wiedemann S, Müller K-R, Samek W (2019) Sparse binary compression: towards distributed deep learning with minimal communication. In: 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, pp 1–8
130. Yu Y, Wu J, Huang L (2019) Double quantization for communication-efficient distributed optimization. Advances in Neural Information Processing Systems 32
131. Li Y, Park J, Alian M, Yuan Y, Qu Z, Pan P, Wang R, Schwing A, Esmaeilzadeh H, Kim NS (2018) A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, pp 175–188
132. Tyurin A, Richtárik P (2024) 2direction: theoretically faster distributed training with bidirectional communication compression. Advances in Neural Information Processing Systems 36
133. Cao X, Başar T, Diggavi S, Eldar YC, Letaief KB, Poor HV, Zhang J (2023) Communication-efficient distributed learning: an overview. IEEE J Sel Areas Commun 41(4):851–873
134. Koloskova A, Loizou N, Boreiri S, Jaggi M, Stich S (2020) A unified theory of decentralized sgd with changing topology and local updates. In: International Conference on Machine Learning. PMLR, pp 5381–5393
135. Abdelmoniem AM, Canini M (2021) Dc2: delay-aware compression control for distributed machine learning. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, pp 1–10
136. Yi X, Zhang S, Yang T, Chai T, Johansson KH (2022) Communication compression for distributed nonconvex optimization. IEEE Trans Autom Control 68(9):5477–5492
137. He Y, Huang X, Yuan K (2024) Unbiased compression saves communication in distributed optimization: when and how much? Advances in Neural Information Processing Systems 36
138. Dai L, Gong L, An Z, Xu Y, Diao B (2024) Sketch-fusion: a gradient compression method with multi-layer fusion for communication-efficient distributed training. J Parallel Distrib Comput 185:104811
139. Minton GT, Price E (2014) Improved concentration bounds for count-sketch. In: Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 669–686

140. Tang H, Gan S, Zhang C, Zhang T, Liu J (2018) Communication compression for decentralized training. Advances in Neural Information Processing Systems 31 (NeurIPS 2018)
141. Koloskova A, Lin T, Stich SU, Jaggi M (2019) Decentralized deep learning with arbitrary communication compression. arXiv preprint arXiv:1907.09356
142. Horváth S, Kovalev D, Mishchenko K, Richtárik P, Stich S (2023) Stochastic distributed learning with gradient quantization and double-variance reduction. Optim Methods Softw 38(1):91–106
143. Rasley J, Rajbhandari S, Ruwase O, He Y (2020) Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 3505–3506
144. Ma T, Hempel M, Peng D, Sharif H (2012) A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems. IEEE Commun Surv Tutor 15(3):963–972
145. Horvóth S, Ho C-Y, Horvath L, Sahu AN, Canini M, Richtárik P (2022) Natural compression for distributed deep learning. In: Mathematical and Scientific Machine Learning. PMLR, pp. 129–141
146. McMahan B, Moore E, Ramage D, Hampson S, Arcas BA (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. PMLR, pp 1273–1282
147. Li Z, Zhao H, Li B, Chi Y (2022) SoteriaFL: a unified framework for private federated learning with communication compression. Adv Neural Inf Process Syst 35:4285–4300
148. Liu Y, Rini S, Salehkalaibar S, Chen J (2023) M22: a communication-efficient algorithm for federated learning inspired by rate-distortion. IEEE Trans Commun 72(2):845–860
149. Wu J, Wang Y, Shen Z, Liu L (2023) Adaptive client and communication optimizations in federated learning. Inf Syst 116:102226
150. Shahid O, Pouriyeh S, Parizi RM, Sheng QZ, Srivastava G, Zhao L (2021) Communication efficiency in federated learning: Achievements and challenges. arXiv preprint arXiv:2107.10996
151. Yang Z, Sun Q (2023) Joint think locally and globally: communication-efficient federated learning with feature-aligned filter selection. Comput Commun 203:119–128
152. Zhu X, Wang J, Chen W, Sato K (2023) Model compression and privacy preserving framework for federated learning. Future Gener Comput Syst 140:376–389
153. Zhao Z, Mao Y, Liu Y, Song L, Ouyang Y, Chen X, Ding W (2023) Towards efficient communications in federated learning: a contemporary survey. J Frankl Inst 360(12):8669–8703
154. Zhou Q, Kousha P, Anthony Q, Shafie Khorassani K, Shafi A, Subramoni H, Panda DK (2022) Accelerating mpi all-to-all communication with online compression on modern GPU clusters. In: International Conference on High Performance Computing. Springer, Berlin, pp 3–25
155. Huang J, Di S, Yu X, Zhai Y, Liu J, Huang Y, Raffenetti K, Zhou H, Zhao K, Chen Z et al (2024) Poster: optimizing collective communications with error-bounded lossy compression for GPU clusters. In: Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, pp 454–456
156. Huang J, Di S, Yu X, Zhai Y, Zhang Z, Liu J, Lu X, Raffenetti K, Zhou H, Zhao K, Chen Z, Cappello F, Guo Y, Thakur R (2024) An optimized error-controlled MPI Collective Framework Integrated with Lossy Compression
157. Yu X, Di S, Zhao K, Tian J, Tao D, Liang X, Cappello F (2022) Ultrafast error-bounded lossy compression for scientific datasets. In: Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, pp 159–171
158. Fei J, Ho C-Y, Sahu AN, Canini M, Sapio A (2021) Efficient sparse collective communication and its application to accelerate distributed deep learning. In: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, pp 676–691
159. Tang H, Gan S, Awan AA, Rajbhandari S, Li C, Lian X, Liu J, Zhang C, He Y (2021) 1-bit adam: Communication efficient large-scale training with adam's convergence speed. In: International Conference on Machine Learning. PMLR, pp 10118–10129
160. Huang J, Di S, Yu X, Zhai Y, Liu J, Huang Y, Raffenetti K, Zhou H, Zhao K, Chen Z et al (2023) GZCCL: compression-accelerated collective communication framework for GPU clusters. arXiv preprint arXiv:2308.05199
161. Huang J, Di S, Yu X, Guo Y. Accelerating collective communications with lossy compression on GPU. Dimensions 449, 849–849235
162. Suresh AT, Felix XY, Kumar S, McMahan HB (2017) Distributed mean estimation with limited communication. In: International Conference on Machine learning. PMLR, pp 3329–3337

163. Liu X, Li Y, Wang R, Tang J, Yan M (2020) Linear convergent decentralized optimization with compression. arXiv preprint arXiv:2007.00232
164. Haneche H, Ouahabi A, Boudraa B (2019) New mobile communication system design for rayleigh environments based on compressed sensing-source coding. IET Commun 13(15):2375–2385
165. Xie H, Qin Z (2020) A lite distributed semantic communication system for internet of things. IEEE J Sel Areas Commun 39(1):142–153
166. Zhang C, Zou H, Lasaulce S, Saad W, Kountouris M, Bennis M (2022) Goal-oriented communications for the IoT and application to data compression. IEEE Intern Things Mag 5(4):58–63
167. Manzhos YS, Sokolova YV (2022) A method of iot information compression. Int J Comput 21(1):100–110
168. Cappello F, Di S, Gok AM (2020) Fulfilling the promises of lossy compression for scientific applications. In: Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI: 17th Smoky Mountains Computational Sciences and Engineering Conference, SMC 2020, Oak Ridge, TN, USA, August 26–28, 2020, Revised Selected Papers 17. Springer, Berlin, pp 99–116
169. Pearlman WA, Islam A, Nagaraj N, Said A (2004) Efficient, low-complexity image coding with a set-partitioning embedded block coder. IEEE Trans Circuits Syst Video Technol 14(11):1219–1235
170. Sakai R, Sasaki D, Nakahashi K (2013) Parallel implementation of large-scale CFD data compression toward aeroacoustic analysis. Comput Fluids 80:116–127
171. Poppick A, Nardi J, Feldman N, Baker AH, Pinard A, Hammerling DM (2020) A statistical analysis of lossily compressed climate model data. Comput Geosci 145:104599
172. Chen Y, Zhang F, Hong Y, Chai Y, Lu W, Chen H, Du X, Wang P, Mi L, Li J et al (2022) Taming the big data monster: managing petabytes of data with multi-model databases. In: 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, pp 283–292
173. Liu Z, Shi Z (2020) A fusion algorithm of multi-model pruning and collaborative distillation learning. In: Journal of Physics: Conference Series, vol 1607, p 012096
174. Shi P, Gao F, Liang S, Yu S (2020) Multi-model inference acceleration on embedded multi-core processors. In: 2020 International Conference on Intelligent Computing and Human–Computer Interaction (ICHCI). IEEE, pp 400–403
175. Bhardwaj K, Lin C-Y, Sartor A, Marculescu R (2019) Memory-and communication-aware model compression for distributed deep learning inference on IoT. ACM Trans Embed Comput Syst TECS) 18(5s):1–22
176. Liu X, Li Y, Tang J, Yan M (2020) A double residual compression algorithm for efficient distributed learning. In: International Conference on Artificial Intelligence and Statistics. PMLR, pp 133–143
177. Ting C, Field R, Fisher A, Bauer T (2018) Compression analytics for classification and anomaly detection within network communication. IEEE Trans Inf Forens Secur 14(5):1366–1376
178. Chen B, Bakhshi A, Batista G, Ng B, Chin T-J (2022) Update compression for deep neural networks on the edge. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 3076–3086
179. Peng J, Fang J, Liu J, Xie M, Dai Y, Yang B, Li S, Wang Z (2023) Optimizing mpi collectives on shared memory multi-cores. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–15
180. Jeon W, Ko G, Lee J, Lee H, Ha D, Ro WW (2021) Deep learning with gpus. In: Advances in Computers. Elsevier, Amsterdam, vol 122, pp 167–215
181. Fadhil A, Haarslev V (2006) Gloo: a graphical query language for owl ontologies. In: OWLED, vol 216
182. Pérez–Acuña B, Fernández-Aller C (2021) Facebook and artificial intelligence: a review of good practices. AI and Ethics 1(3):421–435
183. Hazelwood K, Bird S, Brooks D, Chintala S, Diril U, Dzhulgakov D, Fawzy M, Jia B, Jia Y, Kalro A et al (2018) Applied machine learning at facebook: a datacenter infrastructure perspective. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, pp 620–629
184. Luo L, West P, Nelson J, Krishnamurthy A, Ceze L (2020) Plink: discovering and exploiting locality for accelerated distributed training on the public cloud. Proc Mach Learn Syst 2:82–97
185. Sergeev A, Del Balso M (2018) Horovod: fast and easy distributed deep learning in tensorflow. arXiv preprint arXiv:1802.05799

186. Farhat A, Ghobadi M Communication patterns in distributed deep learning. Massachusetts institute of technology. https://people.csail.mit.edu/ghobadi/theses/amir_farhat_superUROP_report
187. Nielsen MA, Chuang IL (2010) Quantum computation and quantum information. Cambridge University Press, Cambridge
188. Romero J, Olson JP, Aspuru-Guzik A (2017) Quantum autoencoders for efficient compression of quantum data. Quantum Sci Technol 2(4):045001
189. Lee S, Park J, Heo J (2018) Improved reconciliation with polar codes in quantum key distribution. arXiv preprint arXiv:1805.05046
190. Xu M, Niyato D, Yang Z, Xiong Z, Kang J, Kim DI, Shen X (2022) Privacy-preserving intelligent resource allocation for federated edge learning in quantum internet. IEEE J Sel Top Signal Process 17(1):142–157
191. Gaeta AL, Lipson M, Kippenberg T (2019) Photonic-chip-based frequency combs. Nat Photon 13(3):158–169
192. Liu C-Y, Kuo E-J, Lin C-HA, Young JG, Chang Y-J, Hsieh M-H, Goan H-S (2024) Quantum-train: rethinking hybrid quantum-classical machine learning in the model compression perspective. arXiv preprint arXiv:2405.11304
193. Bartolucci S, Birchall P, Bombin H, Cable H, Dawson C, Gimeno-Segovia M, Johnston E, Kieling K, Nickerson N, Pant M et al (2023) Fusion-based quantum computation. Nat Commun 14(1):912
194. Parny L, Alibart O, Debaud J, Gressani S, Lagarrigue A, Martin A, Metrat A, Schiavon M, Troisi T, Diamanti E et al (2023) Satellite-based quantum information networks: use cases, architecture, and roadmap. Commun Phys 6(1):12
195. Preskill J (2018) Quantum computing in the nisq era and beyond. Quantum 2:79
196. Bagherian M, Chehade S, Whitney B, Passian A (2023) Classical and quantum compression for edge computing: the ubiquitous data dimensionality reduction. Computing 105(7):1419–1465
197. George D, Huerta EA (2018) Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. Phys Lett B 778:64–70
198. Dunjko V, Briegel H (2017) Machine learning and artificial intelligence in the quantum domain. arXiv preprint arXiv:1709.02779
199. Luo W, Cao L, Shi Y, Wan L, Zhang H, Li S, Chen G, Li Y, Li S, Wang Y et al (2023) Recent progress in quantum photonic chips for quantum communication and internet. Light Sci Appl 12(1), 175

Authors and Affiliations

Aiqiang Yang[1] · Jie Liu[1,2,3] · Bo Yang[1,2] · Zeyao Mo[4] · Keqin Li[5]

✉ Aiqiang Yang
yangaiqiang0644@nudt.edu.cn

Jie Liu
liujie@nudt.edu.cn

Bo Yang
yangbo78@nudt.edu.cn

Zeyao Mo
zeyao_mo@iapcm.ac.cn

Keqin Li
lik@newpaltz.edu

[1] College of Computer Science and Technology, National University of Defense Technology, Changsha 410000, China

[2] Laboratory of Digitizing Software for Frontier Equipment, National University of Defense Technology, Changsha 410073, China

[3] National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China

[4] China Academy of Engineering Physics, Beijing, China

[5] Department of Computer Science, State University of New York, New Paltz 12561, NY, USA