Gradient Compression Techniques in Distributed DNNs Training Systems: A Survey and Outlook

Zhangqiang Ming, John Doe, *Fellow, OSA,* and Jane Doe, *Life Fellow, IEEE*

Abstract—Frequent communication among computational nodes in distributed model training systems has become one of the major bottlenecks in distributed training. For this reason, many recent studies have utilized gradient compression techniques to reduce communications traffic and distributed training to a large extent. Therefore, gradient compression and its optimization techniques have become a critical component of distributed model training systems and are used widely in the industry. We comprehensively survey gradient compression techniques in distributed model training systems. Firstly, we introduce the background knowledge of the distributed model training system at the system level, including distributed SGD, parallel training schemes, gradient synchronization algorithms, communication system architecture, network communication bottleneck, and communication optimization strategies. Secondly, we focus on combing gradient compression techniques in recently distributed model training systems from the algorithmic level. We classify these techniques into three categories: sparsification compression, quantization compression, and low-rank compression. We subdivide each category further to discuss and analyze the basic principles and advantages/disadvantages. Then, we discuss the error feedback techniques based on memory compensation to optimize gradient compression and collate the current mainstream distributed model training systems. Finally, we summarize the challenges of distributed model training systems at the present phase and provide an outlook on potential future research directions. We help researchers efficiently learn about gradient compression techniques in distributed model training systems research by thoroughly reviewing the existing literature.

Index Terms—Distributed model training systems; Distributed SGD; Parallel training schemes; Gradient synchronization algorithms; Communication system architecture; Network communication bottlenecks; Communication optimization strategies; Gradient compression; Sparsification; Quantification; Low-rank; Memory compensation; Error-feedback.

1 Introduction

Deep neural networks (DNNs) have facilitated the development of artificial intelligence in recent years. To get better inference ability, the parameter size of deep neural networks is getting larger and larger, such as the standard ResNet-152 [1] has about 60.2 MB of parameters, VGG-19 [2] has about 143 MB of parameters, and BERT-Large [3] has a 24-layer network and about 340 MB of parameters. The recent GPT-3 [4] reaches a 96-layer network and about 700GB (175 billion parameters). In deep learning, increasing the size of the training dataset and the complexity of the model often improves the performance of the model. As the dataset's size and the model's complexity increase, the traditional single-computer training process becomes very time-consuming. Therefore, much current research uses distributed training that includes multiple nodes to accelerate the training process, and popular deep learning toolkits, including PyTorch [5], TensorFlow [6], and MXNet [7], support data parallelized training [8]. In distributed training based on data parallelism, each node iterates on its data partition in parallel and exchanges a large number of gradients with other nodes in each iteration through gradient synchronization strategies such as Parameter-Server or Ring-All-Reduce.

However, the frequent gradient swapping of multiple nodes makes the communication take up a large portion of the overall training time [9]. For example, when training Bert-Large [3] and Transformer [10] on 16 AWS EC2 instances, each containing 8 V100 GPUs with a network bandwidth of 100 Gbps, their gradient communication time accounted for 76.8% and 60.9% of the total training time, respectively. Even in a smaller bandwidth network environment, communication accounted for more than 90% [11], which significantly undermines the advantages of distributed training. Therefore, network communication has become one of the main bottlenecks of distributed model training systems [12], [13], [14]. To alleviate the above problem, many researchers have proposed gradient compression during communication to reduce the amount of transmitted data. For example, 1-bit quantization implemented by 1-bit compression algorithm [15] can reduce the amount of transmitted data by 96.9%. Several studies [16], [17], [18], [19], [20] demonstrated through theory and experiments that these compression methods have a limited impact on model convergence and even negligible damage on accuracy. Several large distributed model training systems have integrated gradient compression techniques, such as Horovod [21] and BytePS [22], which are being actively adopted by the industry (e.g., Facebook and AWS).

We classify the gradient compression techniques in distributed model training systems into three main categories: sparsification compression, quantization compression, and low-rank compression. Sparsification compression reduces the amount of communication by selecting the critical information in the gradient tensor for transmission. Quantization compression reduces the communication volume by reducing the number of bits per element in the gradient tensor. On the other hand, low-rank compression methods reduce the amount of communication mainly by performing a low-rank decomposition of the gradient matrix and transmitting the

- *The authors are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. E-mail: {zqming}@hust.edu.cn.*
- *J. Doe and J. Doe are with Anonymous University. E-mail: {zqming}@hust.edu.cn.*

Manuscript received April 19, 2005; revised August 26, 2015.

decomposed submatrix. However, sparsification or quantization compression reduces the accuracy of the gradient, leading to difficulties in model convergence and loss of accuracy [19], [23], [24]. We should pay attention to matrix decomposition's computational time overhead in low-rank compression methods [25]. In response to the problems of gradient compression, more and more researchers tend to optimize it using memory-compensated error feedback-based techniques to compensate for the corresponding performance loss. In addition, some researchers have attempted to design and develop distributed model training systems that can integrate more gradient compression, error feedback, and other optimization techniques, further contributing to developing large-scale models for communication-constrained environments.

Several survey articles have recently summarized and concluded gradient compression techniques for distributed model training [26], [27], [28], but these approaches either focus on overall optimization methods for distributed model training system communication or do not analyze gradient compression techniques in depth. In this paper, we focus on the application of gradient compression as a specific technique in distributed model training systems. In particular, this paper also discusses and analyzes in-depth gradient compression and its error feedback optimization technique and distributed model training systems. However, the above two dimensions were not or rarely mentioned in previous surveys. Therefore, this paper better reflects the current research status and development trend of gradient compression techniques in distributed model training systems and facilitates the reader to understand the field quickly and to conduct subsequent research. To the best of our knowledge, this is the first paper that provides comprehensive survey gradient compression techniques in distributed model training systems.

Specifically, our main contributions are as follows:

- We dissect the background knowledge included in the distributed model training system at the system level and detail the related critical fundamentals, including distributed SGD, parallel training Schemes, gradient synchronization algorithms, communication system architecture, network communication bottlenecks, and communication optimization strategies (reduced parameter update frequency, communication and computation overlap, and gradient compression).
- We comprehensively classify existing gradient compression techniques in distributed model training systems regarding sparsification, quantization, and low-rank compression, which corresponds to the critical sections described in this review. We further break down each class of methods in-depth and provide a detailed analysis and discussion of each class's key ideas, advantages, and disadvantages. This careful classification aims to provide researchers with guidelines for selecting gradient compression techniques and to facilitate subsequent comparative studies.
- We provide an in-depth analysis of the error-feedback schemes for the optimization of gradient compression techniques using memory compensation and classify the error feedback techniques in gradient compression into three significant categories of error accumulation, momentum correction, and error average according to the different forms of memory compensation, and provide a detailed analysis of the methods in each category. In addition, we compile the current recent distributed model training systems, which is beneficial for researchers to select the most suitable system as a benchmark and start subsequent research after understanding the essential characteristics of the system.
- We summarize the remaining challenges of current gradient compression techniques in distributed model training systems. And we also highlight some potential, promising, and worthy directions to explore in the future, including adaptive gradient compression algorithms, reducing the timeshare of gradient decompression, better compression and communication combinations, and unified framework and evaluation metrics for distributed model training systems. All these are beneficial to facilitate distributed model training systems in future research.

Fig. 1 shows the main structure of this paper, which is organized as follows: Section 2 introduces the background knowledge of distributed model training systems, including distributed SGD, parallel training schemes, gradient synchronization algorithms, communication system architecture, network communication bottlenecks, and communication optimization strategies. Next, Section 3 classifies and discusses some gradient compression techniques in distributed model training systems. Section 4 summarizes the current error-feedback methods for gradient compression optimization. Then, we compile the current state-of-the-art distributed model training systems in Section 5. Finally, Section 6 summarizes this thesis and analyzes the challenges and potential future research work of current distributed model training systems.

2 BACKGROUND

This section introduces the background knowledge of distributed model training systems: distributed SGD, parallel training schemes, gradient synchronization algorithms, communication system architecture, network communication bottlenecks, and communication optimization strategies to facilitate readers' understanding of the content of subsequent sections.

2.1 Distributed SGD

Stochastic Gradient Descent (SGD) [29] and its many variants [30], [31] are currently the dominant model training optimizers in machine learning. SGD is a first-order iterative optimization algorithm. At the $(t+1)$-th iteration of the model, SGD updates the model parameter section denoted as:

$$g_t = \nabla f_t(x_t, \xi_t) \tag{1}$$

$$x_{t+1} = x_t - \eta_t \cdot g_t \tag{2}$$

where ξ_t denotes the mini-batch data loaded by the model in the t-th iteration, $x_t \in \mathbb{R}^N$ is the N-dimensional model parameter in the t-th iteration, η_t is the learning rate, and g_t denotes the stochastic gradient in the t-th iteration. Fig. 2 shows the training process of the DNN. We can consider the iterative process of SGD as having three main steps: 1) The model samples the mini-batch data; 2) Performs the feedforward task to calculate the loss function $\nabla f_t(x_t, \xi_t) : \mathbb{R}^d \rightarrow \mathbb{R}$ of the objective function; and 3) Performs the backpropagation task to calculate the gradient value g of each network layer and update the network parameters $x_{t+1} = x_t - \eta_t \cdot g_t$.

The more the model and dataset come, the more time intensive it tends to be to train a deep model. Therefore training using distributed models has become increasingly common. We can

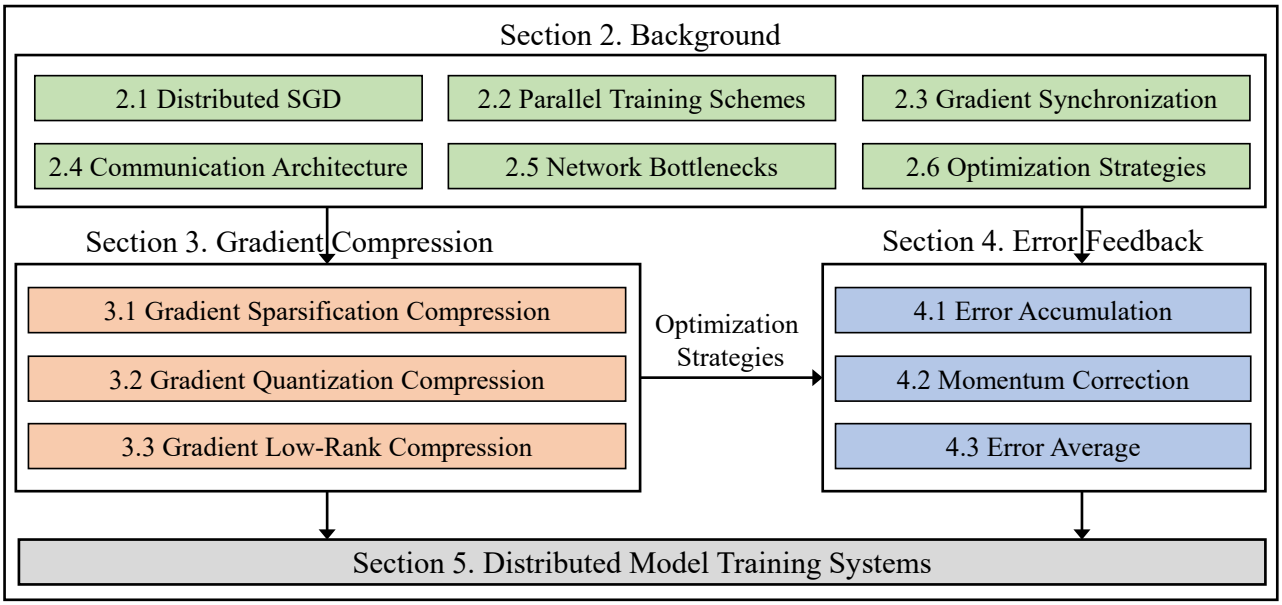


Fig. 1: The structure of this paper

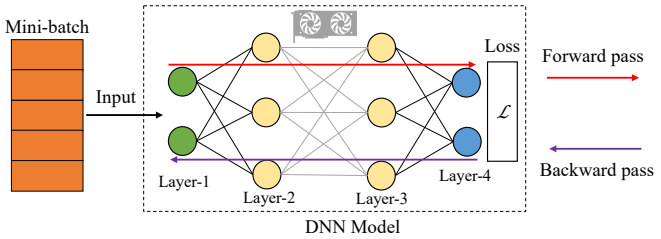


Fig. 2: Schematic diagram of DNN training process

express counterpart bulk synchronous parallel SGD (BSP SGD) as:

$$g_{i,t} = \nabla f_{i,t}(x_{i,t}, \xi_{i,t}) \tag{3}$$

$$x_{i,t+1} = x_{i,t} - \eta_{i,t} \frac{1}{K} \sum_{i=1}^{K} g_{i,t} \tag{4}$$

where $\xi_{i,t}$ denotes the mini-batch data loaded by the i-th Worker in the t-th iteration, $x_{i,t} \in \mathbb{R}^N$ is the $N-$dimensional model parameter. $\eta_{i,t}$ is the learning rate, and $g_{i,t}$ denotes the stochastic gradient $\nabla f_{i,t}(x_{i,t}, \xi_{i,t})$ in the i-th iteration. To speed up the convergence of the model, researchers usually introduce momentum in SGD to update the parameters by using the exponentially weighted average of the gradients instead of the original. For example, we can express the parameter update after the introduction of momentum in Nesterov SGD [32] as:

$$v_{i,t+1} = m \cdot v_{i,t} + \frac{1}{K} \sum_{i=1}^{K} g_{i,t} \tag{5}$$

$$x_{i,t+1} = x_{i,t} - \eta_{i,t}(m \cdot v_{i,t+1} + g_{i,t}) \tag{6}$$

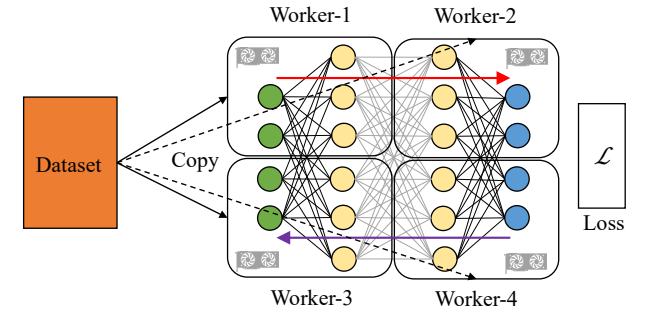
where $0 \leq m \leq 1$ denotes the momentum factor, $v_{i,t+1}$ denotes the temporary parameter, and the gradient $x_{i,t+1}$ of $(t+1)$-th iteration is calculated with the temporary parameters. The difference between distributed SGD and stand-alone SGD is that each worker of distributed SGD computes the gradient independently and sends the gradient for each network layer. The server aggregates and averages gradients from each worker. The worker receives the averaged gradients to update the parameters of their respective models.

2.2 Parallel Training Schemes

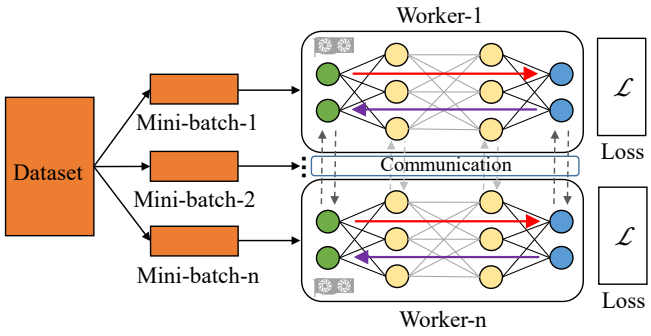
Model and data parallelism are two parallel training schemes commonly used in distributed deep learning. Model parallelism training can be adopted when the trained model is too large to be stored on a single machine. Model parallelism slices the network model and assigns it to different workers, each of which holds other parameters or layers of the model. Each worker collaborates to complete the training of the model. Model parallelism can also be further classified depending on the unit of slicing the network model. One type of model parallelism cuts the network model vertically, i.e., by layer. For example, the deep neural network model assigns different layers to different working nodes responsible for only the corresponding part of the network model structure. Each node completes the data computation and delivery process according to the computational order of the layers assigned in the network model. This model parallelism approach is also known as pipeline parallelism.Another type of model parallelism is the horizontal splitting of the network model, which allows neural network layers containing many parameters to be split on different machines, with the structure shown in Fig. 3 (a). The main advantage of model parallelism is that each worker holds a subset of the model, allowing the training of large models and reducing the memory requirements for individual workers [27].

However, model parallelism still suffers from problems such as unbalanced parameter sizes and strong computational dependency on different layers of the depth model. Therefore, before slicing the network model, it is necessary to consider the network model's connection structure and computational demand and assign them to different computational nodes appropriately, which is one of the difficulties of using model parallelism training [33].

Another parallelism scheme of distributed model training is data parallelism, which means that the training samples are sliced and distributed to different workers for training. As shown in Fig. 3 (b), each worker has a complete copy of the model and uses



(a) Model parallelism



(b) Data parallelism

Fig. 3: Two common parallel schemes for distributed model training.

the assigned data samples to train the model locally, processing different mini-batch data to compute local gradient updates. After the worker computes the local gradient, they communicate with other workers to obtain the global one. Then they use the global one to update the local parameters of the model. We formulate the optimization problem using data parallelism techniques as follows:

$$\min_{x_i \in \mathbb{R}^d} f(x_i) := \frac{1}{K} \sum_{i=1}^{K} E_{\zeta_i \sim D_i} \left[f_{\zeta_i}(x_i) \right] \tag{7}$$

where $x_i \in \mathbb{R}^d$ denotes the parameters of the model, K denotes the number of workers, $E_{\zeta_i \sim D_i} \left[f_{\zeta_i}(x_i) \right] : \mathbb{R}^d \to \mathbb{R}$ represents the loss function of the model parameter x_i on the samples ζ_i in worker i, and D_i indicates the distribution of the training samples in worker i. Data parallelism has theoretical guarantees compared to model parallelism and can achieve the same convergence effect as single-machine single-card training. Data parallelism does not require consideration of model slicing, and parallel implementation is relatively simple, which has become the mainstream distributed training scheme.

2.3 Gradient Synchronization Algorithms

Many workers are involved in the data parallel distributed training. Each worker calculates the gradients locally and then communicates with other workers to obtain the calculation results of other workers and finally get the global one. In this process, the gradient synchronization algorithm decides the frequency of communication among the workers and the timing of the global gradient update. We classify the gradient synchronization algorithms into three categories: synchronous SGD, asynchronous SGD, and local SGD.

Synchronous SGD. Synchronous SGD [34] means that, before starting the next iteration, each worker needs to wait for all other workers in the cluster to finish their computation tasks before updating the global gradients. Synchronous SGD introduces a synchronization barrier technique to ensure that each worker has finished updating the model parameters before the start of a new iteration. The distributed training process using the synchronous SGD technique is similar to training a single batch of data on a single machine, and thus synchronous SGD has theoretical convergence guarantees [35]. Synchronous SGD also has some limitations, which have a high degree of coupling, so when the computation speed of a worker in the cluster is slow, it becomes a bottleneck for the whole distributed training cluster, leading to a waste of resources of idle workers.

Asynchronous SGD. To address the problem of excessive node dependency in synchronous SGD, asynchronous SGD [36] proposes a solution. Each worker gets the model parameter information from the parameter server to train the model independently and then sends the local gradients to the parameter server. The parameter server updates the global one as soon as it receives the local one and does not need to wait for other workers to send gradient information. Asynchronous SGD can reduce the coupling between worker nodes, eliminate the delay caused by the synchronization barrier, and speed up the training speed. However, model training using asynchronous SGD usually suffers from training instability and low accuracy [35].

Local SGD. Based on the synchronous SGD, Local SGD [37] proposes a new gradient synchronization strategy. The main idea of Local SGD is to reduce the frequency of communication between workers. The workers in the cluster update the model parameters directly using the local gradients instead of performing gradient synchronization communication after each iteration. The global gradients are obtained by communication between the workers only when the number of iterations is a predefined particular value. Local SGD can be formulated as follows:

$$x_{i,t+1} = \begin{cases} x_{i,t} - \eta_{i,t} \cdot \bigtriangledown f(x_{i,t}), & \text{if } t+1 \notin \mathcal{S}_T \\ x_{i,t} - \eta_{i,t} \frac{1}{K} \sum_{i=1}^{K} \bigtriangledown f(x_{i,t}), & \text{if } t+1 \in \mathcal{S}_T \end{cases} \tag{8}$$

where $x_{i,t}$ denotes the model parameters of the worker i at the t-th iteration. $\eta_{i,t}$ represents the learning rate, K represents the total number of nodes, $\bigtriangledown f(\cdot)$ denotes the gradient value computed at this time, and $\mathcal{S}_T$ is the predefined set of iteration sequence numbers used for gradient synchronization. Using Local SGD as the gradient synchronization method enables each node to perform multiple iterations locally before global gradient synchronization, which can effectively save communication overhead.

2.4 Communication System Architecture

There are three typical distributed model training gradient synchronous communication architectures: Parameter-Server (PS), All-Reduce (AR), and Ring-All-Reduce (RAR).

Parameter-Server (PS): As shown in Fig. 4 (a), there are two roles in the Parameter-Server (PS) architecture, server and worker. The server is responsible for keeping the latest global model and collecting updates from the worker. The worker loads the dataset independently, extracts the latest global gradient from the server, computes the updated parameters, and transmits them to the server. The server is the center of the communication topology in this architecture, so researchers also refer to PS as a centralized framework. It is widely used in distributed training [38], [39].

The main challenge of the PS architecture is the communication congestion of the server because all workers are communicating with the server.

All-Reduce (AR): To avoid the communication bottleneck in PS, researchers tend to use the All-Reduce architecture [34], [40], [41], [42], [43] to implement gradient aggregation. In this architecture, each device (node) has its local model and loads parts of the dataset separately. Nodes train a model jointly by minimizing the loss function on their corresponding dataset. To ensure that the parameters of each node's model are updated synchronously, the nodes communicate with each other in a peer-to-peer manner. As shown in Fig. 4 (b), this method has only one role, i.e., worker. Each worker sends messages to and receives information from all other workers. They get all gradients of all workers and then update their local model. Thus, AllReduce is a decentralized communication topology and a centralized model topology because of the consistent global gradient obtained by synchronization.

Ring-All-Reduce (RAR): This strategy uses a collective communication primitive. A representative example is Ring-All-Reduce [44], where all nodes are workers, which form a logical ring (Gossip). As shown in Fig. 4 (c), RAR consists of two communication steps, the first step is to aggregate the gradients along the ring, and the other step is to propagate the updated gradients. In addition, Ring-All-Reduce can perform batch processing of the gradients and then partition the gradients for load balance. After that, in each synchronization step, each worker sends a partition to its successor worker simultaneously and receives another partition from its predecessor to utilize its bidirectional network bandwidth better. All workers communicate without a central server, so researchers refer to RAR as a decentralized framework.

2.5 Network Communication Bottlenecks

The time overhead of distributed training consists of computation and communication overhead, where each worker must frequently communicate to exchange their local gradients to update the local model parameters. However, the performance bottleneck in distributed training is increasingly shifting from computation to communication. To explore the network bottleneck in distributed training, Sapio et al. [14] trained eight standard DNN benchmark models using NVIDIA P100 GPUs on a cluster containing eight workers and calculated the communication time over 10 Gbps or 100 Gbps Ethernet. Fig. 5 shows the above experimental results, where "10 Gbps Comm" is the proportion of communication time for gradient synchronization to the total model training time in 10 Gbps Ethernet, and "10 Gbps Overlap" indicates the percentage of communication time overlapping with the computation time. It shows five DNN benchmarks with more than 50% communication time in a 10 Gbps network environment, among which DeepLight's communication time accounts for 97%, and only 2% overlaps with the computation. In the training of DeepLight, The communication used for gradient synchronization takes up most of the training time, and the overlap between computation and communication is negligible. Even in a 100 Gbps network environment, communication time accounts for 79% of the total training time. In contrast, the overlap between communication and computation is only 20%, indicating that the current network is saturated and network communication is a severe bottleneck for distributed training.

In addition, Bai et al. [9] trained two popular DNN frameworks, Bert-large and Transformer, on 16 AWS EC2 clusters, where each instance contained eight NVIDIA V100 GPUS with a network bandwidth of 100 Gps. Without gradient compression, the communication time accounts for 76.8% and 60.9% of the total training time, respectively. It shows that a significant portion of the gradient communication time does not overlap with the computation time in the DNN computation. Gradient compression is a popular method to reduce the amount of transmitted data during gradient communication [18], [20] and has great potential to solve the communication bottleneck problem [9]. For example, the classical gradient quantization algorithm 1-bit SGD [15] can reduce the amount of transmitted gradient data by 96.9%, significantly reducing the overhead communication time.

In summary, network communication is still the main bottleneck in data-parallel DNN training with limited bandwidth. There is a fundamental conflict between communication and computation. The communication time for gradient synchronization accounts for a large proportion of the overall model training time, which is more prominent on nodes with low network bandwidth. It is often impractical to reduce communication overhead by increasing network bandwidth, so researchers prefer to use gradient compression techniques to reduce communication overhead.

2.6 Communication Optimization Strategies

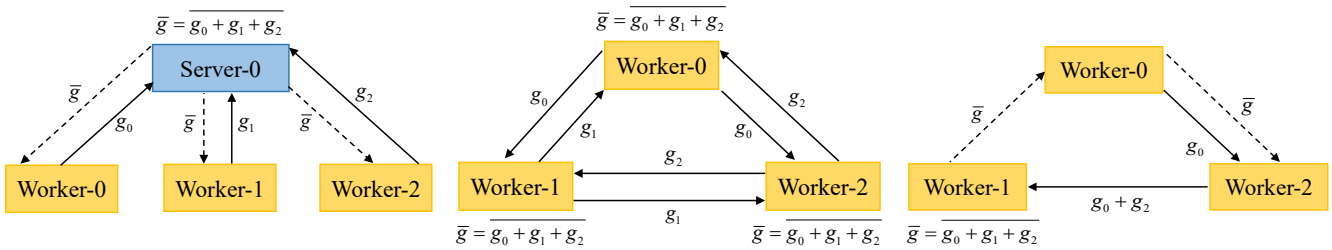
The primary techniques used to reduce the communication overhead of distributed training include: reducing the frequency of parameter updates, increasing the overlap between communication and computation, and gradient compression techniques. Some schemes combine multiple optimization techniques, such as gradient compression and overlapping techniques, that can be applied simultaneously to distributed training and work together to reduce the time overhead. The following section will describe these three optimization techniques' main ideas and their representative methods.

2.6.1 Reducing the Frequency of Parameter Updates

In distributed training, the workers exchange gradients in each iteration to update their parameters. Hence, an intuitive way to reduce communication overhead is to reduce the frequency of parameter updates. There are two main improvement options: 1) increase the Batch-size to reduce the number of model iterations and 2) update the gradient periodically to reduce the number of gradient exchanges.

Batch size is a significant hyperparameter for model training, determining the amount of loaded data in each iteration. Therefore, for a given data set size, Batch-size determines the number of iterations and the number of gradient exchanges. When the Batch-size is larger, the number of model iterations is smaller, and the model parameters are updated less frequently, thus reducing the communication overhead due to the less frequent gradient exchange. However, a large Batch-size means a large amount of GPU memory. In addition, because increasing the Batch-size causes the model to converge to the local minima of the training function [34], [45], the model will have poor model generalization ability. Several studies have proposed including warmup strategy [34], [46], tensor offloading [47], and layerwise adaptive scaling [48], [49], [50] to solve this problem. For example, You et al. [48] proposed a new layerwise adaptation strategy called LAMB. LAMB enables adaptive adjustment of SGD learning rate and effectively saves time when training BERT [3] and ResNet-50 [1].

Frequent gradient synchronization is another crucial reason for the high communication time overhead of distributed training.



(a) Parameter-Server (PS) (b) All-Reduce (AR) (c) Ring-All-Reduce (RAR)

Fig. 4: Gradient synchronous communication architecture for distributed model training

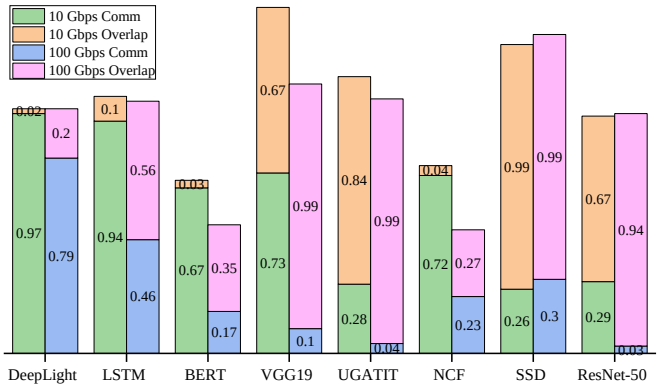
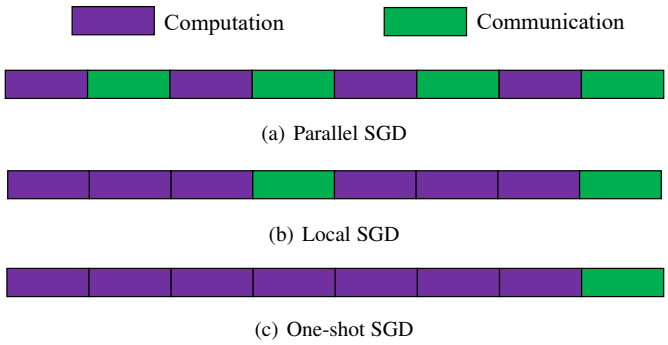


Fig. 5: Communication and overlap time percentages for eight benchmark models trained on 10 Gbps and 100 Gbps networks, respectively.



(a) Parallel SGD

(b) Local SGD

(c) One-shot SGD

Fig. 6: Three different SGD gradient update strategies.

Fig. 6 (a) shows the gradient synchronization strategy of vanilla parallel SGD. This strategy communicates immediately after computing each model layer's gradients, significantly increasing the time overhead. Therefore, some researchers have considered periodic averaging of the model parameters trained on local nodes, which can reduce the frequency of gradient synchronization and the communication overhead of training [37], [51], [52], [53], [54], [55], [56], [57]. Fig. 6 (b-c) show two examples of gradient updates using periodic parameter averaging. In addition, some researchers have demonstrated that an appropriate period of parameter averaging does not affect training convergence [37], [53], [58], [59].

2.6.2 Increasing the Overlap Between Communication and Computation

In the backward propagation, the gradients are generated sequentially from the last layer to the first layer of the model. During communication, workers send the gradients of each layer to the server or other workers separately and receive the average gradients after aggregation. There is an overlap between the computation and communication of the gradients, i.e., the gradients generated first by the later model layer can be sent first and do not affect the generation and communication of the gradients in the former layer [60]. Ideally, the worker can send the gradients of the L-th layer while computing the gradients of the $(L-1)$-th layer without waiting to generate all the gradients before sending them. This scheme prevents the network from blocking and saves the total training time (computation + communication) by reducing the waiting time. Therefore, the communication and computation tasks can be described as a directed acyclic graph (DAG) for pipeline scheduling [61]. Many studies about parallel execution of computation and communication have recently emerged [13], [41], [60], [62], [63], [64], [65], [66], [67], mainly to optimize the execution order of computation and communication at each layer of the model.

For example, some wait-free-back-propagation (WFBP) scheduling algorithms [41], [61], [62], [68] draw on the idea of FIFO. These algorithms overlap computation and communication by synchronizing each layer in the network independently, requiring workers to send the gradient of each layer immediately after calculating the gradient, as shown in Fig. 7 (b). Since different layers may have different computation and communication times, delay terms (startup time) can cause communication delays. For this reason, Shi et al. [60] proposed merged gradient WFBP (MG-WFBP) to mitigate the negative impact caused by startup time. Some studies [13], [63], [64], [65] reorganize communication tasks and computation tasks by scheduling the execution order or partitioning the tensor, the general idea of which is shown in Fig. 7 (c). Peng et al. [13] propose a general communication scheduler called ByteScheduler. ByteScheduler achieves better overlap between computation and communication by partitioning and reorganizing tensor transfers. Jayarajan et al. [64] implemented a similar scheduling strategy on MXNet-PS with a higher training speedup than the FIFO approach. In addition, some methods [24], [69], [70], [71] consider the computation time of gradient compression and implement pipeline parallelism between communication and gradient compression computation.

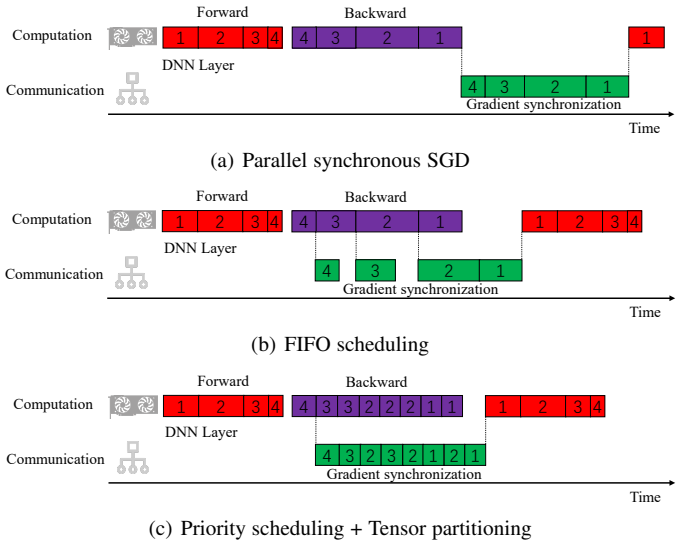


Fig. 7: Communication overhead timeline for three scheduling strategies.

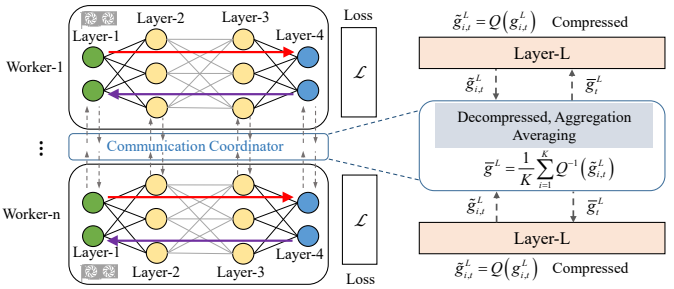


Fig. 8: Schematic diagram of the gradient compression process of the L-th layer of DNN.

2.6.3 Gradient Compression

Gradient compression can reduce the amount of transmitted data. It has been widely used in distributed training to reduce communication costs [23]. The gradient compression technique mainly consists of two processes, compression and decompression, and can be defined as follows:

$$\tilde{g}_{i,t}^{L} = Q(g_{i,t}^{L}) \tag{9}$$

$$\overline{g}_{t}^{L} = \frac{1}{K}\sum_{i=1}^{K} Q^{-1}(\tilde{g}_{i,t}^{L}) \tag{10}$$

where $g_{i,t}^{L}$ represents the gradient of the L-th layer of the model in the i-th worker at the t-th iteration, Q represents the compressor, $\tilde{g}_{i,t}^{L}$ represents the compressed gradient, Q^{-1} represents the decompressor, and $\overline{g}_{t}^{L}$ represents the average gradient of the L-th layer in all workers at the t-th iteration. As shown in Fig. 8, in backward propagation, each worker calculates the gradient $g_{i,t}^{L}$ of the L-th layer and sends the compressed gradients $\tilde{g}_{i,t}^{L}$ to the receiver (server or other workers). After receiving the compressed gradients, the receiver decompresses them using $Q^{-1}(\tilde{g}_{i,t}^{L})$ and averages them after aggregation. After that, each worker receives the averaged gradient $\overline{g}_{t}^{L}$ and updates the parameters. If the expectation value satisfies $E(Q(x)) = x$ for compressor Q, then Q belongs to unbiased compression. Otherwise, it belongs to biased compression [8].

In this paper, we classify the gradient compression techniques into three significant categories: gradient quantization, gradient sparsification, and low-rank compression. We will summarize and analyze the gradient compression algorithms in detail in Section 3. Most compression methods are lossy, causing poor model convergence or low accuracy. Some studies introduce error feedback (or memory compensation) mechanisms to alleviate these problems. We will discuss the specific details and common methods of error feedback in Section 4.

3 Gradient Compression Techniques

We classify gradient compression techniques into three major categories: *Sparsification compression* [16], [17], [18], [69], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], *Quantization compression* [11], [15], [19], [20], [87], [88], [89], [90], [91], [92], [93], and *Low-rank compression* [25], [94], [95], [96], [97]. We further subdivide the above classification into random sparsification and deterministic sparsification for sparsification compression, deterministic and random quantization for quantization compression, and centrality and decentralism for low-ranking compression methods. Fig. 9 shows the specific classification of gradient compression techniques.

3.1 Sparsification Compression

Gradient sparsification compression is a promising gradient compression technique applied to distributed SGD. Its main idea is to select a subset of gradient elements for communication to reduce the communication overhead significantly. Gradient sparsification is more flexible, can reduce communication overhead exponentially, and is easily adaptable to various network conditions [69]. Results have shown that in some cases, gradients can be sparse by up to three orders of magnitude to zero while ensuring model convergence [16], [17], [18], [75], [76], [77]. In this paper, we classify sparsification compression into two main categories: stochastic sparsification and deterministic sparsification.

3.1.1 Stochastic Sparsification

The main idea of stochastic sparsification is to randomly select a subset of the gradients to communicate and update to reduce communication overhead. Several implementations of stochastic sparsification have been proposed [72], [74], [84], [85]. For example, Konecny et al. [84] proposed two stochastic sparsification methods, random mask and random sampling, to improve the efficiency of upstream communication. Stich et al. [72] first analyzed the convergence of Random-k by randomly selecting k indices, transmitting only the gradients corresponding to these k indices, and then multiplying the sparsified gradient vector by a scaling factor to achieve an unbiased update. Considering that the variance of the gradients would affect the convergence speed, Wangni et al. [74] proposed an unbiased stochastic sparsification method to maximize the sparsity and control the variance. The core idea of this method is to randomly remove some coordinates of the gradient vector and properly scale up the remaining coordinates to ensure the unbiasedness of the sparsification gradients. In addition, a similar approach has been proposed by Tsuzuku et al. [85].

3.1.2 Deterministic Sparsification

Unlike stochastic sparsification, deterministic sparsification selects deterministic gradients for communication updates by introducing

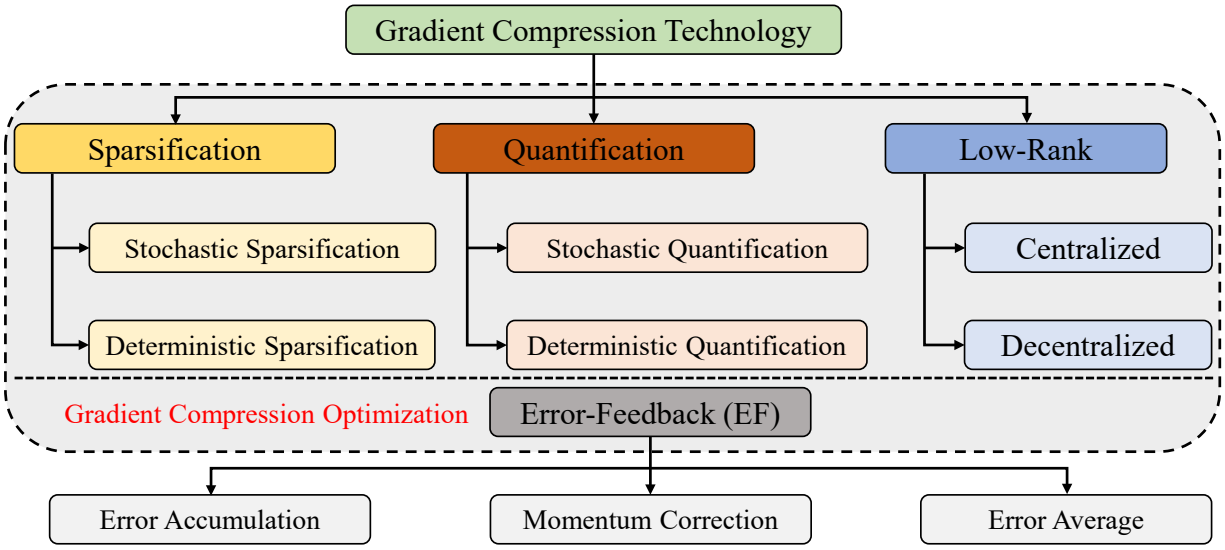


Fig. 9: Classification of gradient compression and error-feedback techniques.

additional computational overhead. Since most of the gradients are around zero [84], i.e., the gradients themselves have sparse properties, selecting significant (e.g., large absolute values) gradients for updating can reduce the communication overhead while ensuring model convergence. Most of the deterministic sparsification methods [16], [17], [18], [69], [73], [75], [76], [77], [78], [79], [80], [81], [82], [83] use Top-k, i.e., each node transmits only the largest k gradient elements where k is a fixed sparsity ratio, usually taken as 1% or 0.1% [16], [18].

Most deterministic sparsification methods use the residual gradient technique for error feedback[16], [17], [18], [78], [81] to ensure accuracy and convergence. In each iteration, workers accumulate the gradients not transmitted at each node locally and add them to the gradient vector in the next iteration for delayed updates. To reduce the computational overhead, most deterministic sparsification methods use a threshold-based approach to achieve approximate Top-k selection [16], [18], [77], [78], [80]. Deterministic sparsification can be further divided into fixed-threshold sparsification [17] and adaptive-threshold sparsification [16], [73], [75], [77], [78], [79], [80], [81], [82].

Fixed-Threshold Sparsification. Fixed-threshold sparsification compression, like other existing compression algorithms [14], [87], belongs to the category of absolute compression, which has an absolute constraint on the error [98]. Strom et al. [17] proposed a sparse method that selects a fixed threshold first and only transmits those gradients whose absolute value is greater than this threshold. Since the subgradients on each node are very sparse, only those gradients that have a high impact on the update of the weights need to be transmitted. In contrast, the gradients close to zero can be delayed and thus reduce communication bandwidth usage. Strom et al. constructed a key-value mapping to encode the obtained sparsification gradients, where the keys are the indices of the complete gradients, and the values are the corresponding individual gradient elements. This method applies the residual gradient technique to guarantee accuracy and convergence, ensuring that weights with small but biased subgradient values will eventually be updated. However, the obvious drawback of fixed-threshold sparsification is that finding a suitable threshold value is difficult. If the threshold value is too high, convergence suffers, and thus accuracy is reduced. If the threshold is too low, a large number of gradients will be transmitted, resulting in slower training.

Adaptive-Threshold Sparsification. To address the drawbacks of fixed-threshold sparsification, Dryden et al. [77] proposed an adaptive threshold sparsification method. It uses a fixed ratio and selects two thresholds, one positive and one negative, in each iteration to ensure the transmission and update the gradient of the fixed percentage. This method combines the advantages of both 1-bit quantization [15] and fixed-threshold sparsification [17] with a trade-off between computational overhead and compression level.

Considering that previous methods are based on a specific model structure, application to other DNNs leads to accuracy loss and poor compression of the convolutional layers during training, especially when compressing fully connected layers of the same model. Chen et al. [78] proposed AdaComp. Based on simple local sampling, AdaComp adaptively selects the degree of compression sparsity according to different neural layers, mini-batch size, and other factors. AdaComp does not require any sorting or near-sorting computation and is computationally efficient for high-performance systems. Since determining two thresholds [77] is too tedious, Aji et al. [16] proposed an alternative adaptive thresholding method that uses the layer normalization (LN) [99] to select the global threshold and filter the Top-k elements based on the absolute value of the gradient. To further compress and ensure accuracy, Lin et al. [18] proposed deep gradient compression (DGC), which reduces the overhead of global Top-k selection by sampling and can achieve a compression rate of 99.9% with little impact on accuracy.

To solve the problem of poor quality of threshold estimation in previous methods [79], [81], Abdelmoniem et al. [80] proposed a sparsity-based distributed compression method called SIDCo, which considered the sparsity distribution property of the gradient. SIDCo models the gradients according to some sparsity distribution [100], [101] and uses a multi-stage fitting method to obtain the threshold value. Compared with DGC [18], RedSync [81], and Gaussian-k [79], SIDCo is faster and has higher model accuracy with less overhead.

Most of the deterministic sparsification methods basically use the gradient size as an indicator to filter the gradient [16], [18], [75], [77]. However, the size of the gradients only indicates

the optimization direction of the current model parameters. It cannot indicate the importance of the parameters, thus may lead to delayed parameter updates. Zhang et al. [82] proposed an adaptive gradient sparsification framework, MIPD. Considering the different significance of the parameters in each neural network layer, MIPD uses the model interpretability (MI) and exponential smoothing prediction (ESP) of the gradient to calculate the corresponding compression ratio by layer. MIPD can adaptively select the significant gradients for transmission to reduce the communication overhead and decrease the gradient error.

3.1.3 Discussion

Table 1 summarizes the ideas and limitations of the gradient sparsification compression method. The critical idea of gradient sparsification is that only significant gradients are needed to update the model parameters. According to the way of selecting gradients, we classify gradient sparsification into stochastic sparsification and deterministic sparsification, represented by Random-k [37] and Top-k [16], respectively. The stochastic sparsification randomly selects a subset of the gradient, optimizes for the deviation and variance of the gradients, and uses the memory compensation to ensure convergence speed while reducing communication overhead. Due to the stochastic nature, selecting the significant gradients for stochastic sparsification is difficult.

The main idea of deterministic sparsification is Top-k, primarily based on selecting adaptive thresholds. Deterministic sparsification mainly optimizes the following aspects: 1) ensuring convergence performance while deepening the sparsification; 2) designing a Top-k selection algorithm suitable for GPUs with low computational overhead; 3) exploiting the probability distribution properties of gradients and 4) layer-wise gradient sparsification. Experimentally, some methods exploit the properties of gradients to design heuristics to estimate and find Top-k thresholds, which have lower compression overhead compared to Top-k and DGC [79], [80], [81], [98].

In summary, gradient sparsification compression reduces the communication overhead in DNN training by selecting a subset of the gradients and transmitting these subsets. The selection of compression gradients may be stochastic or deterministic. The main ways to improve the gradient sparsification compression are as follows: 1) Most existing methods are for specific communication architectures, such as AR and PS. It is a challenge to adopt appropriate sparsification methods in different communication architectures. 2) Current methods mostly use gradient size as the only indicator for gradient sparsification, but gradient size cannot indicate the importance of the corresponding parameters. Therefore, identifying the significant gradients and assigning them higher update priority is a worthy research direction.

3.2 Quantization Compression

Gradient quantization truncates or maps the gradient elements to a predefined set of discrete values, allowing the gradient to be represented in training with fewer bits, thus reducing the impact of gradient communication on training speed.

In this paper, we classify gradient quantization algorithms into two categories: deterministic quantization [15], [87], [88], [89], [91] and stochastic quantization [11], [19], [20], [92], [93], [102]. Deterministic quantization usually adopts an aggressive quantization strategy, but there will be a particular case where training fails to converge. In contrast, stochastic gradient quantization methods have theoretical convergence guarantees.

3.2.1 Deterministic Quantization

The deterministic quantization method means that the quantized gradient element values are always deterministic for a given gradient. Seide et al. [15] proposed 1-bit SGD in 2014: using only 1 bit to represent the gradient element values. 1-bit SGD quantizes the gradient elements smaller than a user-defined threshold (0 by default) as '0' and the remaining elements as '1'. At decompression, '0' is replaced by the average of negative elements, and '1' is replaced by the average of non-negative elements. They also proposed an error feedback method: the worker would locally record the difference between the quantized gradient and the original gradient, which will be added to the quantization computation in the next iteration. Strom et al. [17] proposed an approach similar to the 1-bit SGD in 2015, which combines gradient sparsification: each worker will only send the quantized value of a specific element and the corresponding index.

An equally aggressive quantization method is the SIGNSGD [89], [91] proposed by Bernstein et al. The main idea is to replace the original gradient elements with the signs of the gradient elements, which can be expressed as follows:

$$x_{i,t+1} = x_{i,t} - \eta_{i,t} \cdot sign(g_{i,t}) \tag{11}$$

where $x_{i,t}$ represents the model weights of worker i at iteration t, $g_{i,t}$ represents the gradient calculated at iteration t, $sign(\cdot)$ represents the sign of the gradient element, and $\eta_{i,t}$ represents the learning rate of the training. SIGNSGD uses a method called Majority Vote at the parameter server nodes, which allows the parameter server to quantize the gradients again after aggregating all the gradients, further reducing the amount of transmitted data. In order to speed up the training of deep neural networks and improve the robustness of SIGNSGD. Bernstein proposed the SIGNUM [89], [91] method based on SIGNSGD. SIGNUM makes the training more stable by introducing momentum in calculating gradient updates. Since SIGNSGD loses the magnitude and direction of the gradient, it has poor generalization ability compared to vanilla SGD and fails to converge in some simple cases [88]. Karimireddy et al. [88] improved SIGNSGD using an error feedback method and proposed EF-SIGNSGD. Karimireddy shows that the error feedback strategy can solve the problem of failure to converge due to biased compression algorithms like SIGNSGD.

Furthermore, Dettmers et al. [87] proposed an 8-bit quantization method to represent 32-bit single-precision floating-point numbers as 8-bit integer numbers. This algorithm uses 1 bit as the sign bit, 3 bits as the exponent bit, and the lowest 4 bits to represent the mantissa, which retains more gradient information compared to 1-bit SGD.

3.2.2 Stochastic Quantization

Stochastic rounding has been shown to perform well in low-precision training [103]. Several studies have introduced stochastic rounding in gradient quantization. The stochastic quantization algorithm makes the quantization results have unbiased expectations compared to biased quantization, which eliminates the quantization error from a statistical point of view and provides a guarantee of convergence in model training. Wen et al. [20] proposed TernGrad in 2017. TernGrad uses a stochastic algorithm to quantize the gradient element value into one of three elements $\{-1, 0, 1\}$, allowing us to represent a 32-bit floating-point number with 2 bits. The quantization is as follows:

$$\tilde{g}_{i,t} = s_{i,t} \cdot sign(g_{i,t}) \circ b_{i,t} \tag{12}$$

TABLE 1: Characteristics and limitations of several sparsification compression methods

Dims	Methods	Ref.	EF	Characteristics	Limitations
Stochastic	Random mask	[84]		Uses a random mask to transmit the sparse update matrix.	Targeted to the federation learning scenario and considers only the network architecture of PS.
	Random-k	[72]	✓	Randomly selects k elements for transmission and ensures convergence speed by memory compensation.	Converges slowly, and the degree of compression is not adaptive.
	Variance-based	[74]		Randomly removes some gradient coordinates and appropriately scales up the remaining coordinates.	Tends to lose some significant gradients leading to model non-convergence.
Deterministic	Fixed Threshold	[17]	✓	Picks a fixed threshold and only transmits and updates elements whose absolute values are more significant than this threshold.	Difficult to choose a suitable threshold.
	Fixed Proportion	[77]	✓	Uses a fixed ratio and selects one positive and one negative threshold to ensure transmission.	Needs to sort the gradient elements with high computational overhead.
	Top-k	[16]	✓	Uses the LN technique [99] to select a global threshold that can filter the Top 1% of elements.	Only targets specific network structures and models.
	AdaComp	[78]	✓	Adaptively adjusts the degree of compression based on different layers, small batch sizes, and other factors.	Introduces new hyperparameters, which are challenging to determine the optimal values in different models.
	DGC	[18]	✓	Reduces the overhead of Top-k selection by sampling to achieve a compression ratio of 99.9%.	Needs to perform Top-k twice in the worst case, with high computational overhead.
	gTop-k	[75]		Selects global Top-k elements from all Workers to reduce the communication overhead.	Doesn't consider the sparsification of the stratification.
	RedSync	[81]	✓	Finds a threshold by adjusting the ratio between the maximum and average values of the gradient to filter out Top-k elements.	Has poor prediction accuracy, and the randomness of the filtered elements is significant.
	Gaussian-k	[79]	✓	Heuristically obtains Top-k elements by fitting a Gaussian distribution to adjust the initial threshold.	Only sometimes satisfies the strong prior condition in meeting the Gaussian distribution, resulting in poor compression quality.
	SIDCo	[80]	✓	Models the gradient based on some predictions and obtains the thresholds by a multi-stage fitting method.	Doesn't consider stratification for gradient sparsification.
	MIPD	[82]	✓	Takes advantage of the model interpretability of the gradient and the exponential smoothing prediction method to filter out the significant elements in a hierarchical manner.	Only for PS architectures with asynchronous updates.

$$s_{i,t} \triangleq \|g_{i,t}\|_{\infty} \triangleq max(abs(g_{i,t})) \tag{13}$$

where $s_{i,t}$ denotes the infinite norm of $g_{i,t}$, i.e., the largest absolute value of each element of the gradient. $\circ$ denotes the Hadamard product. $sign(\cdot)$ and $abs(\cdot)$ represent each element's sign and absolute value, respectively. $b_{i,t}$ represents a random binary vector, and each element of $b_{i,t}$ independently obeys to the Bernoulli distribution.

TernGrad can achieve 16× compression of gradients. To reduce the errors caused by gradient quantization, TernGrad proposes *layer-wise ternarizing* and *gradient clipping* in training. QSGD [11] proposed by Alistarh et al. is a work of the same period as TernGrad and employs stochastic rounding to ensure unbiased gradient quantization compression. QSGD algorithm can be expressed as follows:

$$Q_s(g_i) = \|g\|_2 \cdot sign(g_i) \cdot \zeta_i(g, s) \tag{14}$$

where $s \geq 1$ is a hyperparameter that controls the level of quantization compression. $\zeta_i(g, s)$ is an independent random variable.

$$\zeta_i(g,s) = \begin{cases} (l+1)/s, & probability\ p(|g_i| \,/\, \|g\|_2, s), \\ l/s & otherwise. \end{cases} \tag{15}$$

where $p(a, s) = as - l$, $a \in [0, 1]$. l represents the interval where the gradient elements are located, where $|g_i| \,/\, \|g\|_2 \in [l/s, (l+1)/s]$ and $0 \leq l < s$. QSGD maps the normalized gradient elements to a uniform set of discrete values $\{0, 1/s, 2/s, \ldots, (s-1/s), 1\}$. By controlling the hyperparameter s, QSGD allows trainers to make a trade-off between gradient accuracy and compression level. Based on QSGD, Wu et al. [19] proposed ECQ-SGD by introducing error feedback. It accumulates all previous quantization errors and passes them to the next iteration. According to empirical evaluation, ECQ-SGD can better suppress the effect of quantization on the upper bound of errors compared to QSGD. Faghri et al. [92] proposed an adaptive version of QSGD called AQSGD. AQSGD dynamically adjusts the distribution of the discrete values l_i during training to minimize the expected value of the gradient variance, speeding up the convergence of the model training.

Yan et al. [102] proposed DQ-SGD in the same year. DQ-SGD is also an adaptive algorithm that dynamically adjusts the number of compressed bits of the QSGD according to the number of iterations and the gradient size during training. Horvóth et al. [93] proposed Natural compression in 2022. The Natural algorithm uses a series of powers of 2 as a set of discrete values and outperforms the QSGD algorithm in terms of training performance.

3.2.3 Discussion

Gradient quantization is one of the mainstream solutions to solve the communication bottleneck problem in distributed training. Representative works of stochastic quantization algorithms are QSGD [11] and TernGrad [20]. Stochastic quantization algorithms have unbiasedness and theoretical convergence guarantees. However, the randomization introduced by unbiased quantization schemes will increase the variance of the quantized gradient. In the worst case, stochastic quantization can even lead to variance explosion [89], severely affecting the convergence rate of model training.

A simple compression idea for deterministic quantization is to reduce the accuracy of the representation of the gradient elements. Although it is usually biased, it has been shown to converge under convexity and sparsity [104]. More aggressive representative

TABLE 2: Characteristics and limitations of several quantization compression methods

Dims	Methods	Ref.	EF	Characteristics	Limitations
Stochastic	TernGrad	[20]		Has an unbiased theoretical compression convergence guarantee and can achieve significant communication savings.	Must be combined with other techniques when applied to large-scale training.
	QSGD	[11]		Has a theoretical convergence guarantee and allows a trade-off between communication efficiency and convergence speed.	Increases the gradient variance, which hinders the convergence of the model.
	ECQ-SGD	[19]	✓	Introduces an error compensation strategy to improve the stability of model convergence.	The introduction of error feedback techniques brings additional computational overhead.
	Natural	[93]	✓	Has less computational overhead.	The maximum compression ratio achieved by the method is limited.
	AQSGD	[92]		Proposes a dynamic selection strategy of quantization parameters to speed up the model convergence.	Dynamically determines the distribution of the discrete values of the quantization, which results in additional computational overhead.
	DQ-SGD	[102]		Adaptively adjusts the number of QSGD compression bits to minimize the communication overhead.	Introduces computational overhead in calculating the number of compression bits and is hard to implement in code.
Deterministic	1-bit SGD	[15]	✓	Has the highest compression ratio and can save communication time.	Imposes computational time overhead for calculating the gradient element average.
	SIGNSGD	[91], [89]		Has the highest compression ratio and can achieve bi-directional compression of data communication.	Has poor generalization ability and doesn't converge in some cases.
	SIGNUM	[91], [89]	✓	Introduces momentum, which makes the model training more stable.	Has poor generalization ability and doesn't converge in some cases.
	EF-SIGNSGD	[88]	✓	Introduces error feedback to make the model convergence more robust.	Error feedback brings additional computational overhead.
	8-bit SGD	[87]		Solves the problem of a large batch size of 1-bit SGD in large-scale training.	The compression ratio of the method has an upper limit and cannot achieve higher quantization ratios.

works of deterministic quantization algorithms are 1-bit SGD [15], SIGNSGD [89], [91], and EF-SIGNSGD [88]. These schemes are usually based on the threshold or the sign of the gradient elements, quantizing the gradient elements to fewer bits. Despite lacking theoretical guarantees of unbiasedness, such aggressive quantization compression methods usually perform much better than unbiased compression [88]. It is worth noting that such biased compression schemes are usually combined with error feedback to obtain the expected desired performance. Table 2 illustrates the mainstream gradient quantization compression techniques' advantages and limitations.

No single gradient quantization scheme has an absolute advantage so far. The best-performing gradient quantization schemes vary depending on the model and the deep-learning task selected. Gradient quantization has the advantage of combining the multi-threading technique to quantize and compress the gradient elements, which can reduce the computational overhead of compression to some extent. However, gradient quantization also has some limitations: the theoretical upper limit of the compression ratio is 32×. Moreover, the gradient quantization technique usually has to be used in conjunction with auxiliary training techniques to get a better training speedup.

3.3 Low-rank Compression

Most deep learning computing is performed in the form of matrices. Real-world deep learning tasks often use millions of magnitudes of data as training samples to improve the accuracy and generalization of models. The exponential growth in the number of parameters and computation of models makes matrix-based gradient operations more frequent and complex, which increases the communication load of distributed model training. Several studies have found that DNNs are over-parameterized while the gradients exhibit a low-ranked structure [94], [105], [106]. Based on this pattern, the low-rank gradient matrix can be decomposed

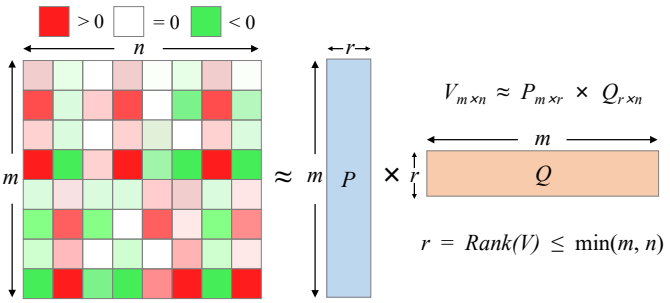


Fig. 10: Schematic diagram of the decomposition process of low-rank gradient matrix

into two submatrix products using low-rank decomposition. The process of gradient decomposition is shown in Fig. 10. Where the gradient matrix $V \in \mathbb{R}^{m\times n}$ exhibits a low-rank structure with a rank of size $r = Rank(V)$, it is decomposed into two submatrices $P \in \mathbb{R}^{m\times r}$ and $Q \in \mathbb{R}^{r\times n}$ products of rank r using matrix decomposition. The low-rank decomposition of a matrix avoids matrix communication by sending smaller vectors.

Although some lossy gradient compression (e.g., quantization, sparsification) methods [11], [20], [88], [89], [91] can alleviate the communication bottleneck in data distributed training, these methods are prone to conflict with collective arithmetic standards (e.g., Ring-All-Reduce) and even lead to model non-convergence. There are three main ideas to solve the above problems [25]: 1) linear compression utilizes the linear correlation between gradients to achieve compatibility with collective arithmetic standards; 2) error feedback mechanisms can compensate for the adverse effects caused by gradient compression and ensure the convergence and accuracy of the model remain unchanged; and 3) low-rank decomposition can achieve high-efficiency compression without sacrificing accuracy. We classify low-rank compression methods

into centralized methods [25], [94], [95] and decentralized methods [96], [97] according to their application scenarios.

3.3.1 Centralized

Existing compression schemes either run slower than SGD with optimized full-decrement gradient aggregation or fail to achieve the same test performance. Wang et al. [94] proposed a general framework for atomic sparsification of stochastic gradients called ATOMO. ATOMO factorizes the gradient matrix to minimize the quantized stochastic gradient variance. ATOMO does this by controlling the gradient variance with a smaller set of primitives representing the random gradients to produce a low-rank approximation of gradients. Experiments show that applying ATOMO's gradient singular value decomposition (SVD) can converge the model faster than Vanilla SGD or QSGD under the same communication conditions. However, ATOMO requires the computation of an expensive SVD step and has poor scalability.

Vogels et al. [25] proposed PowerSGD, which uses a generalized power iteration to compute a low-rank approximation to the gradient, decomposing the original matrix into a product of two submatrices of the same rank. PowerSGD has the characteristics of linear compression, the low computational cost of matrix low-rank decomposition, and allows All-Reduce gradient aggregation while having the full precision SGD test performance. In addition, PowerSGD uses momentum correction to avoid introducing additional hyperparameters. Since the current collective communication standard MPI is usually defined to handle dense data representations, the sparse tensor after gradient sparsification compression does not apply to AR operations. Therefore, Cho et al. [95] proposed GradZip, which, unlike popular sparsification-based compression techniques, uses the results of the low-rank decomposition of the previous synchronization to provide a high-quality gradient decomposition matrix for the current synchronization.

The essence of low-rank matrix decomposition is the output of a dense representation of compressed gradients capable of communication and statute operations via AR, the collective computing standard in most high-performance deep learning frameworks.

3.3.2 Decentralized

Lossy gradient compression has emerged as a practical tool to overcome communication bottlenecks in centrally coordinated distributed training, which can be performed with a significant reduction in communication while retaining complete model accuracy. However, algorithms for decentralized training with compressed touch on arbitrarily connected networks have been more complex and usually require additional memory and hyperparameters. Low-rank decomposition performs well in centralized deep learning [25], [28], [95] and achieves the same experimental performance in decentralized training as compression methods with quantization or sparsification compression [96], [97].

Yu et al. [97] first proved that CNN gradients are strongly linearly correlated with each other and proposed a novel gradient compression technique called GradiVeQ. GradiVeQ uses Principal Component Analysis (PCA) to obtain the correlation between the gradients and decomposes them to achieve compressed dimensionality reduction. In addition, GradiVeQ also performs aggregation operations on the compressed directly without decompressing them, realizing the complete parallelism of gradient compression and RAR communication, thus significantly reducing the communication load in distributed CNN training.

Inspired by the centralized deep learning PowerSGD algorithm [25] and GradZip algorithm [95]. Vogels et al. [96] proposed a low-rank compression technique for decentralized distributed model training called PowerGossip. PowerGossip sets the model's parameters as a matrix and decomposes each pair of adjacent nodes in a low-ranked approximation to estimate the parameter differences between the nodes. By compressing the differences in the model, PowerGossip can achieve the low-rank approximation without passing the whole matrix, which ensures model accuracy and significantly reduces communication between workers.

3.3.3 Discussion

Sparsification or quantization compression methods tend to cause non-convergence of the model or loss of model accuracy. The low-rank compression decomposes the matrix based on the low-rank characteristics exhibited by the gradient. It avoids matrix communication by sending smaller vectors, which does not affect the convergence of the model. Table 3 shows several methods of low-rank compression. Among the centralized methods represented are ATOMO [94], PowerSGD [25], and GradZip [95]. However, these methods cannot directly aggregate the compressed gradients, thus incurring an extended compression time overhead, which does not apply to decentralized aggregation protocols.

Decentralized low-rank compression techniques can alleviate the above problems, focusing on low-rank approximations of the gradients between neighboring nodes. PowerGossip [96], for example, is an improvement on PowerSGD and GradZip that only requires tuning the learning rate in a more straightforward centralized setup and then transitioning to decentralized learning, in contrast to previous decentralized approaches that required algorithm modifications and additional hyperparameters [55], [107], [108]. PowerGossip does not require additional memory and hyperparameters.

The disadvantages of the low-rank compression method still exist in that the decomposition process takes up many computational resources and additional time overhead. These adverse effects usually cannot be ignored in distributed model training. Moreover, since different gradient matrices exhibit different low-rank situations, using only one decomposition method does not guarantee the best results in the subsequent synchronization process. Therefore, low-rank compression techniques have yet to be applied heavily in distributed model systems.

3.4 Discussion

In addition to the above three compression methods, some hybrid methods combine quantization and sparsification compression techniques [17], [77], [109], [110], [111], which are not discussed here in this paper. Most of the gradient sparsification and quantization compression methods belong to lossy compression. Usually, the higher the degree of gradient compression, the lower the communication delay due to gradient swapping. At the same time, it also causes the loss of gradient accuracy and the convergence speed of the model [11].

Furthermore, we cannot ignore the computational overhead introduced in training by sparsification or quantization compression, which increases the overall iteration time if the overhead is greater than the reduction gain in communication time [79], [81], [112]. In gradient low-rank compression approaches, matrix decomposition often results in substantial computational and time overheads. And the large variety of gradient generation in

TABLE 3: Characteristics and limitations of several low-rank compression methods

Dimension	Methods	Ref.	Characteristics	Limitations
Centralized	ATOMO	[94]	Controls the gradient variance by controlling the gradient variance and representing the random gradient with a set of fewer primitives.	The decomposition using SVD has large computational and time overheads and lacks scalability.
	PowerSGD	[25]	The method is computationally inexpensive and has the performance of a full precision SGD test.	The tests mainly focus on small datasets, and the effect is not apparent on extensive models and datasets.
	GradZip	[95]	The dense vectors obtained from the low-rank decomposition communicate and reduce directly using All-Reduce.	Alternating low-rank decomposition causes additional computational overhead.
Decentralized	GradiVeQ	[97]	Uses PCA to decompose the gradient and realize the gradient dimensionality reduction.	Only applicable to the case of a strong linear correlation between CNN gradients.
	PowerGossip	[96]	No extra hyperparameters are required, and convergence is faster, making it a plug-and-play compression algorithm.	Vulnerable to data heterogeneity.

different datasets in different models makes it difficult for low-rank compression approaches to trade off their own computational and communication overheads. Therefore, low-rank compression methods are not widely used in practice. Fig. 11 shows a visual comparison of several compression schemes mentioned above.

There are two central optimization schemes for gradient compression. One of the optimization schemes is to design adaptive and optimal compression algorithms. Some recent studies [113], [114], [115], [116], [117], [118], [102], [119] take full advantage of gradient compression. For example, Accordion [117] adjusts the communication scheduling dynamically by adaptively checking the gradient change when the training is in a critical state [120]. Accordion applied to Top-k and PowerSGD reduces the communication and increases the convergence speed with no loss of accuracy or change of hyperparameters. Another optimization scheme introduces error feedback, which feeds back the error loss caused by compression to the next iteration. The error feedback technique for gradient compression is described and analyzed in detail in Section 4 of this paper.

4 Error Feedback Technique

Although gradient compression reduces the communication time of gradient exchange to some extent, there are still problems. These gradient compression methods are lossy compression methods of gradients, which can easily affect the accuracy of model training or even lead to difficulties in model convergence. Therefore, many researchers have used error-feedback (EF) or memory to compensate for the loss of accuracy. EF was first proposed by [15], [17] in gradient compression and subsequently [72], [88], [120] analyzed the convergence of the EF framework and was widely used for distributed model training [24], [120]. Some researchers [72], [73], [88] record the difference between the original and compressed gradients in memory by error accumulation and compensate for it in the next iteration. Some researchers [18], [121], [122], [123] have also used momentum correction techniques to mitigate the error loss due to compression after gradient accumulation. Other researchers [23] have significantly reduced the communication overhead by averaging the compression errors, requiring only the errors of the local parameters to be compressed and transmitted. Depending on the form of memory compensation, we classify the methods of error feedback in gradient compression into three categories: error accumulation, momentum correction, and error averaging. Fig. 9 demonstrates the details of the specific classification of error feedback techniques.

4.1 Error Accumulation

Most of the gradient compression methods are lossy or biased, which leads to poor accuracy and even failure to converge the model [15]. Therefore, some researchers [19], [72], [73], [88] considered adding the error between the compressed gradient and the original gradient to the gradient of the next iteration to achieve the accumulation of the compressed gradient error and proved to be able to improve the convergence speed and guarantee the accuracy of the model without affecting the communication time overhead. For example, Karimireddy et al. [88] proposed EF-SIGNSGD to save the compression error locally and add it to the next step to achieve the accumulation of the compression gradient error and obtain the same convergence rate of SGD. Their iterative loop steps of SGD based on error accumulation are summarized as follows: 1) add the error to the next step to achieve accumulated error; 2) compress the gradient of accumulated error and update the parameters; and 3) calculate the error between the actual gradient and the compressed gradient and save it locally. Eqs. (16, 17, 18,19) represent the above corresponding steps. Where $e_{i,t}$ denotes the error of the current compressed gradient, $p_{i,t}$ denotes the gradient after SGD introduces the cumulative error, $\eta_{i,t}$ is the learning rate, $Q(\cdot)$ and $Q^{-1}(\cdot)$ denote the compression and decompression functions, respectively. $\tilde{g}_{i,t}$ is the compressed gradient.

$$p_{i,t} = \eta_{i,t} g_{i,t} + e_{i,t} \tag{16}$$

$$\tilde{g}_{i,t} = Q(p_{i,t}) \tag{17}$$

$$x_{i,t+1} = x_{i,t} + \frac{1}{K} \sum_{i=1}^{K} Q^{-1}(\tilde{g}_{i,t}) \tag{18}$$

$$e_{i,t+1} = p_{i,t} - \tilde{g}_{i,t} \tag{19}$$

There are also some error accumulation-based methods [16], [19], [77], [78], [85], [88], [111], [120], [121], [122] with similar design ideas to EF-SIGNSGD. Stich et al. [120] further developed EF-SIGNSGD and generalized the EF framework with improved rates. In addition, Wu et al. [19] considered the compression error in the current iteration and all the gradient errors in the historical iterations to accumulate the compression error in each iteration.

4.2 Momentum Correction

Although error accumulation can mitigate the problem of model convergence and accuracy degradation caused by gradient compression, the accumulated historical errors can occupy a large amount of memory and lead to unsynchronized node parameter updates. Therefore, some researchers [18], [121], [122], [123]

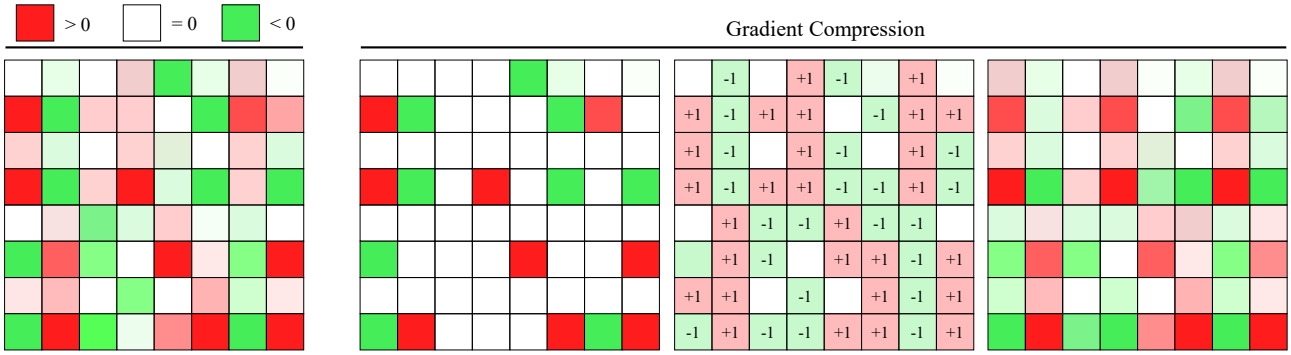


Fig. 11: Visual comparison of three compression schemes

considered momentum correction to mitigate the adverse effects caused by gradient compression. Some research works consider reducing the frequency of parameter updates to avoid the communication time overhead from frequent gradient swapping, i.e., transmission only when the worker accumulates local gradients to a large enough size. The specific implementation process of accumulating gradients can be found in Section 2, Subsection 2.6 of this paper.

However, as time passes, each worker eventually sends all the gradients. When the gradient sparsity is high, the frequency of gradient updates increases sharply, making it difficult for the model to converge to the minimum. As in Fig. 12 (a), the gradient accumulation causes the model to converge only to point *C*, while the model doesn't converge to the original optimal point *B*.

To address the above problem, Lin et al. [18] proposed a deep gradient compression (DGC) technique, which uses momentum correction and local gradient clipping to maintain the performance of the model based on gradient sparsification compression. The momentum correction of DGC can be expressed as:

$$u_{i,t+1} = m \cdot u_{i,t-1} + g_{i,t} \tag{20}$$

$$v_{i,t} = v_{i,t-1} + u_{i,t} \tag{21}$$

$$\tilde{g}_{i,t} = Q(v_{i,t}) \tag{22}$$

$$x_{i,t+1} = x_{i,t} + \eta_{i,t} \frac{1}{K} \sum_{i=1}^{K} Q^{-1}(\tilde{g}_{i,t}) \tag{23}$$

where m denotes the momentum factor, $u_{i,t}$ denotes the momentum at the t-th iteration, and $v_{i,t}$ denotes the cumulative velocity after momentum correction. The first two items (Equ. 20, 21) are the local gradient accumulation after momentum correction. The accumulation velocity $v_{i,t}$ is used for the subsequent compression and communication. The momentum correction ensures that the update of the sparsification gradient is equivalent to the dense update, as shown in Fig. 12 (b). In addition, the momentum correction is only an adjustment of the update equation and does not produce any hyperparameters. Zhao et al. [124] argued that DGC lacks a theoretical convergence analysis and proposed a new technique called global momentum compression (GMC). GMC also combines cumulative gradient and momentum SGD and theoretically proves the convergence rate of GMC for convex and non-convex functions.

4.3 Error Average

SignSGD [91], [89] lacks generalization capability, and convergence is problematic mainly due to two reasons: 1) loss of gradient values and 2) loss of gradient directions. Recently, researchers have introduced local error feedback methods [88] to fix the corresponding non-negligible performance loss of SignSGD by adding the compression error of the current iteration to the next iteration. Local error feedback adding the current compression error to the next iteration may cause the gradient of the next iteration to be outdated. For this reason, Xu et al. [23] analyzed the new "gradient mismatch" problem caused by local error feedback, which may lead to stale gradients and may not achieve lossless performance, proposed step-ahead error feedback (SAEF). The primary expression of SAEF is as follows:

$$\bar{e}_{i,t} = \frac{1}{K} \sum_{i=1}^{K} e_{i,t} \tag{24}$$

$$x_{i,t+\frac{1}{2}} = x_{i,t} - \bar{e}_{i,t} \tag{25}$$

$$g_{i,t+\frac{1}{2}} = \bigtriangledown f(x_{i,t+\frac{1}{2}}; \xi_{i,t}) \tag{26}$$

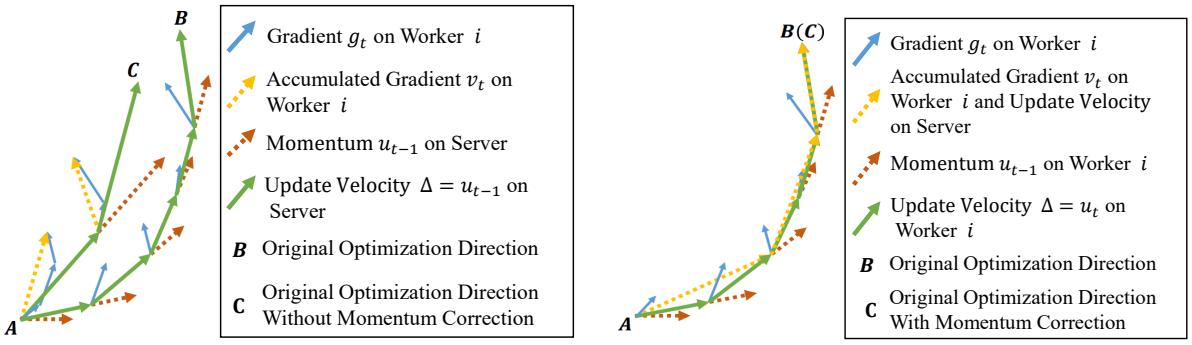
$$m_{i,t+1} = \mu_{i,t} \cdot m_{i,t} + g_{i,t+\frac{1}{2}} \tag{27}$$

$$x_{i,t+1} = x_{i,t+\frac{1}{2}} + \eta_{i,t} \cdot m_{i,t+1} \tag{28}$$

$$\triangle_{i,t+1} = \bar{e}_{i,t} + x_{i,t+\frac{1}{2}} - x_{i,t+1} \tag{29}$$

$$e_{i,t+1} = \triangle_{i,t+1} - Q(\triangle_{i,t+1}) \tag{30}$$

where $\bar{e}_{i,t}$ denotes the average compression error of the t-th iteration, $x_{i,t+\frac{1}{2}}$ represents the parameter updated one step ahead, $g_{i,t+\frac{1}{2}}$ represents the gradient updated one step ahead, and $m_{i,t+1}$ represents the momentum term. SAEF updates the parameter $x_{i,t+1}$ of the next step using the parameter $x_{i,t+\frac{1}{2}}$ and the momentum term $m_{i,t+1}$ to obtain the local parameter error $\triangle_{i,t+1}$ of the next step, which is compressed and then sent to other nodes. $e_{i,t+1}$ represents the compressed error. The step-ahead parameter update can alleviate the problem of "gradient mismatch" problem and ensure timeliness. Compared with the compressed gradient method, the error averaging method only requires the compression and transmission of the local parameters errors, which can vastly reduce the amount of data transmission. Moreover, SAEF can achieve better convergence bounds with no performance loss compared to the general methods of full precision training and local error feedback [20], [88], [89], [109].



(a) Local gradient accumulation without momentum correction (b) Local gradient accumulation with momentum correction

Fig. 12: Comparison schematic of momentum correction [18]

4.4 Discussion

EF is one of the popular methods to improve model convergence and gradient compression generalization [88]. The methods of error accumulation, momentum correction, and error average all facilitate model convergence and prevent accuracy loss, but they also differ significantly in their design ideas. Error accumulation saves the errors of the original gradient and gradient compression locally and passes them on to the next iteration, requiring only an increase in local memory and a small amount of computation. The error accumulation method still passes the compressed gradient information in synchronous communication, and this method can better compensate for the loss caused by the gradient compression.

The error averaging method passes the compression gradient errors instead of the compressed gradient information during synchronous communication and stores the average of the compression errors of each node locally. In addition, the error averaging method can significantly reduce communication overhead compared with the error accumulation method. It is vulnerable to the compression errors of other nodes and does not guarantee that the gradients of each layer of the network are updated to the same level synchronously. The momentum correction method differs from the previous two methods in that it reduces the frequency of parameter updates by accumulating the gradients locally until they are large enough to be transmitted. The methods of momentum correction mainly use the momentum accumulation rate to correct the local parameters so that the model can converge to the original optimal value.

5 Distributed Model Training Systems

Fig. 13 shows the overall architecture of the distributed model training system, which consists of four main components: 1) Gradient synchronization; 2) Compression techniques; 3) Optimization strategies; and 4) Communication architecture. These four components have been described in detail in the previous sections of this paper, and most of the current research work is focused on these four components.

Some deep learning frameworks come with API for distributed model training, such as PyTorch [5], TensorFlow [6], and MXNet [7]. However, they have limited functionality and do not support the optimization and scaling of large-scale distributed model training. Recently, some high-performance distributed training frameworks such as Horovod [21] and BytePS [22] have integrated the above deep learning frameworks to support large-scale distributed and simultaneous training on a cluster of GPU resources, which are easy to use, portable, efficient, and reliable. Therefore, many research works [8], [9], [13], [81], [125], [126] are based on the above-distributed training frameworks and design distributed model training systems with different features for different application scenarios.

For example, Xu et al. [8] proposed a generic compressive communication framework for distributed model training called Grace. Grace integrates TensorFlow and PyTorch, implements 16 representative compression methods, and exposes the necessary functionality for various compressive communication methods such as compression, decompression, and memory compensation in the form of API. In addition, Grace tested convolutional neural networks (CNNs) and recurrent neural networks (RNNs) on datasets from three different domains (e.g., datasets image classification and segmentation, recommender systems, and language modeling) and evaluated metrics including accuracy, throughput, and communication, respectively, so that Grace can serve as a benchmark for researchers to implement and evaluate new methods. Bai et al. [9] proposed a compression-aware-based data-parallel DNN training framework, called HiPress. HiPress is compatible with mainstream DNN systems (MXNet, TensorFlow, and PyTorch) and achieves high-performance pipelined parallelism of computation and communication through flexible recombination of computation and communication primitives. In addition, HiPress integrates five advanced compression algorithms (1-bit [15], TBQ [17], TernGrad [20], DGC [18], and GradDrop [16]) into the DNN system. On three DNN models for computer vision and three natural language processing, HiPress improved the training speed by 17.2%-69.5% over current benchmark compressed systems such as BytePS and Horovod. Song et al. [127] proposed a distributed graph neural network (DGNN) with gradient compression and error feedback based on PyG [128], called EC-Graph. EC-Graph can greatly reduce the computational and communication costs between training nodes by message compression.

Table 4 gives an overview of several recent distributed model training systems. grace [8] mainly implements the integration and evaluation of multiple compression algorithms and analyzes various impact metrics of gradient compression algorithms. HiPress [9] similarly integrates multiple algorithms and proposes a compatible strategy for gradient compression algorithms and gradient synchronization to alleviate the conflict between performance improvement and programming overhead. Therefore, Grace does

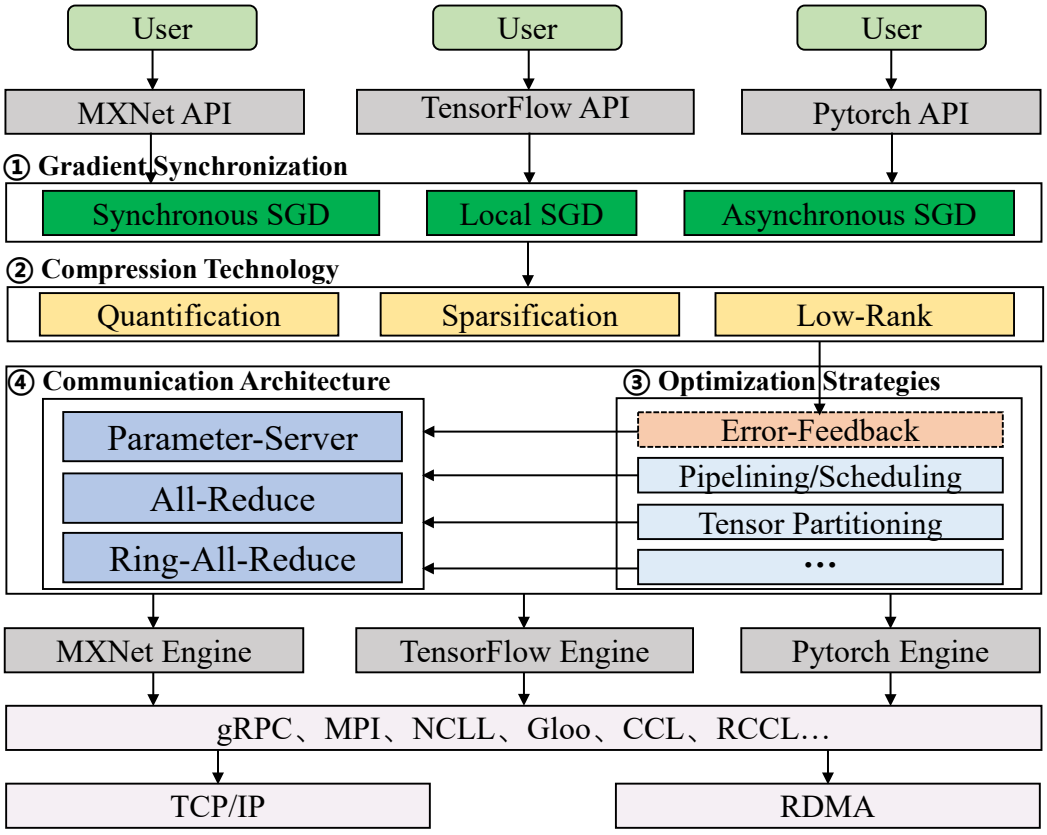


Fig. 13: The schematic diagram of distributed model training system architecture

TABLE 4: Overview of several recent distributed model training systems

Framework	Ref.	Benchmark	Compression methods	EF	Main Work
Grace	[8]	Horovod	Quantification, Sparsification	√	Implemented 16 compression algorithms.
HiPress	[9]	Horovod, BytePS	Quantification, Sparsification	√	Implemented 5 compression algorithms and provided a unified API.
CGX	[125]	Horovod	Quantification, Sparsification		Seamless, parameter-free integration with mainstream frameworks.
Bagua	[126]	PyTorch, TensorFlow	Quantification	√	The modular design of communication algorithms.
RedSync	[81]	Horovod	Quantification		Support for sparse process and efficient system deployment.
ByteScheduler	[13]	BytePS			Generalized tensor scheduling framework.

not analyze and address the system-level challenges compared to HiPress. Bagua [126] modularizes the communication set to make the system more flexible and efficient. CGX [125] uses NCCL [129] as the communication backend to achieve higher communication efficiency than HiPress and has a better compression ratio than Bagua. ByteScheduler [13] focuses on communication optimization for gradient tensor optimal scheduling. In general, many of the current compression algorithms or other optimization strategies have not been developed, optimized, and integrated into distributed model training systems, which puts a heavy burden on DNN practitioners and is not conducive to the grounded application of the methods. Therefore, designing more efficient, easy-to-operate, and scalable distributed model training systems remains the focus of future research.

6 Conclusion and Future Research Work

We comprehensively survey gradient compression techniques in distributed model training systems. We first present the background of the distributed model training systems, including distributed SGD, parallel training schemes, gradient synchronization algorithms, communication system architecture, network communication bottlenecks, and communication optimization strategies. The purpose is to provide the reader with an understanding of the process of distributed model training. Then, we summarize the gradient compression techniques in the distributed training system and divide them into sparsification, quantization, and low-rank compression. We thoroughly analyze the design ideas and advantages and disadvantages of each category of compression methods. In addition, we dissect the error feedback scheme optimized for gradient compression techniques and introduce the latest distributed model training systems. Finally, we summarize the existing challenges in distributed model training systems and provide exciting and practical future research directions for optimizing distributed model training systems.

Although there have been a large number of studies related to distributed model training systems, they still face the following challenges: 1) Gradient compression is flawed, and traditional compression methods either cause model accuracy degradation and convergence difficulties or introduce optimization measures such as error feedback to increase memory consumption and training complexity; 2) The time overhead of compression and

decompression is not negligible and may be greatly amplified in the gradient synchronization process is greatly amplified, making some compression methods inapplicable in practice; 3) There is a lack of coordination between the computation (compression) and communication of gradients, and there is still a large overlap time between computation and communication, which is more prominent for low-bandwidth networks; and 4) The existing distributed model training systems are poorly integrated and lack scalability. Integrating various existing optimization algorithms into a unified training framework becomes very difficult. Based on the above challenges, we look forward to some interesting future research directions:

Adaptive Gradient Compression Algorithms. For different datasets, the size, and sparsity of the gradients generated by various network layers vary relatively widely, and traditional compression algorithms are often not applicable. In the future, we will consider adaptively setting different compression rates or strategies at different network layers or upstream and downstream network bandwidths to achieve efficient network communication without affecting the accuracy and convergence of the model.

Aware Gradient Decompression Time. The current study lacks a detailed analysis of the time overhead caused by gradient decompression. However, this time overhead cannot be ignored and may significantly impact the overall communication time during subsequent gradient synchronization. Therefore, in the future, we will first analyze the compression time of different compression algorithms for different data sets or network environments as a percentage of the total communication time and then optimize the compression scheme by combining this factor.

Efficient Compression and Communication Combinations. The lack of coordination between the computation (compression) and communication of gradients often results in network blocking or incomplete utilization of network resources. In the future, we will consider decoupling the gradient compression and communication primitives to achieve a more optimal combination of compression and communication or communication scheduling strategy. In addition, the gradients can be partitioned, and the fine-grained gradient tensor facilitates increased overlap of gradient compression and communication, thus enabling efficient pipelining of compression and communication tasks.

Unified Framework for Distributed Training Systems. Although most existing compression or communication optimization methods are designed based on existing distributed model training frameworks such as Horovod and BytePS. Different algorithms or optimization methods are independent of each other and lack a unified and standardized API interface to integrate them into a distributed model training system. Therefore, it is crucial to design a distributed model training system with a unified framework, programming API, and quantitative evaluation metrics in the future.

ACKNOWLEDGEMENT

The authors wish to thank Jizhuo Li, Xiao Pang, Jiamin Zhu, Fuqiu Chen, Yi Zhou and Cheng Zhang. This work was supported by the National Key Research and Development Project of China (No. JG2018190), and in part by the National Natural Science Foundation of China (No. 61872256).

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, IEEE, 2016.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, and A. Davis, "Tensorflow: a system for large-scale machine learning," in *Osdi*, vol. 16, pp. 265–283, Savannah, GA, USA, 2016.

[7] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[8] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "Grace: A compressed communication framework for distributed machine learning," in *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*, pp. 561–572, IEEE, 2021.

[9] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu, "Gradient compression supercharged high-performance data parallel dnn training," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 359–375, 2021.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017.

[11] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, pp. 8024–8035, 2017.

[12] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, "Parameter hub: a rack-scale parameter server for distributed deep neural network training," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 41–54, 2018.

[13] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 16–29, 2019.

[14] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[15] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "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*, September 2014.

[16] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[17] N. Ström, "Scalable distributed dnn training using commodity gpu cloud computing," in *Interspeech 2015*, 2015.

[18] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[19] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized sgd and its applications to large-scale distributed optimization," in *International Conference on Machine Learning*, pp. 5325–5333, PMLR, 2018.

[20] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *Advances in neural information processing systems*, vol. 30, pp. 1508–1518, 2017.

[21] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[22] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters," in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pp. 463–479, 2020.

[23] A. Xu, Z. Huo, and H. Huang, "Step-ahead error feedback for distributed training with compressed gradient," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 10478–10486, 2021.

[24] S. Zheng, Z. Huang, and J. Kwok, "Communication-efficient distributed blockwise momentum sgd with error-feedback," *Advances in Neural Information Processing Systems*, vol. 32, pp. 11446–11456, 2019.

[25] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Powersgd: Practical low-rank gradient compression for distributed optimization," *Advances in Neural Information Processing Systems*, vol. 32, pp. 14236–14245, 2019.

[26] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, "Communication optimization strategies for distributed deep neural network training: A survey," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 52–65, 2021.

[27] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," *arXiv preprint arXiv:2003.06307*, 2020.

[28] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "Compressed communication for distributed deep learning: Survey and quantitative evaluation," 2020.

[29] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[30] E. Moulines and F. Bach, "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," *Advances in neural information processing systems*, vol. 24, p. 451–459, 2011.

[31] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pp. 177–186, Springer, 2010.

[32] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical programming*, vol. 140, no. 1, pp. 125–161, 2013.

[33] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, p. 1223–1231, 2012.

[34] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[35] K. S. Chahal, M. S. Grover, K. Dey, and R. R. Shah, "A hitchhiker's guide on distributed training of deep neural networks," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 65–76, 2020.

[36] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," *Advances in neural information processing systems*, vol. 23, p. 2595–2603, 2010.

[37] S. U. Stich, "Local sgd converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.

[38] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, p. 19–27, 2014.

[39] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang, *et al.*, "Singa: A distributed deep learning platform," in *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 685–688, 2015.

[40] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.

[41] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-caffe: Co-designing mpi runtimes and caffe for scalable deep learning on modern gpu clusters," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 193–205, 2017.

[42] C.-H. Chu, X. Lu, A. A. Awan, H. Subramoni, J. Hashmi, B. Elton, and D. K. Panda, "Efficient and scalable multi-source streaming broadcast on gpu clusters for deep learning," in *2017 46th International Conference on Parallel Processing (ICPP)*, pp. 161–170, IEEE, 2017.

[43] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, "Impact of network topology on the performance of dml: Theoretical analysis and practical factors," in *IEEE INFOCOM 2019-IEEE conference on computer communications*, pp. 1729–1737, IEEE, 2019.

[44] A. Gibiansky, "Bringing hpc techniques to deep learning," *Baidu Research, Tech. Rep.*, 2017.

[45] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[46] Y. You, J. Hseu, C. Ying, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large-batch training for lstm and beyond," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2019.

[47] J. Bae, J. Lee, Y. Jin, S. Son, S. Kim, H. Jang, T. J. Ham, and J. W. Lee, "Flashneuron: Ssd-enabled large-batch training of very deep neural networks," in *FAST*, pp. 387–401, 2021.

[48] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.

[49] Z. Huo, B. Gu, and H. Huang, "Large batch optimization for deep learning using new complete layer-wise adaptive rate scaling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7883–7890, 2021.

[50] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Fast deep neural network training on distributed systems and cloud tpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2449–2462, 2019.

[51] H. Su and H. Chen, "Experiments on parallel training of deep neural network using model averaging," *arXiv preprint arXiv:1507.01239*, 2015.

[52] J. Zhang, C. De Sa, I. Mitliagkas, and C. Ré, "Parallel sgd: When does averaging help?," *arXiv preprint arXiv:1606.07365*, 2016.

[53] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5693–5700, 2019.

[54] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *2016 ieee international conference on acoustics, speech and signal processing (icassp)*, pp. 5880–5884, IEEE, 2016.

[55] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," *Advances in Neural Information Processing Systems*, vol. 31, p. 7663–7673, 2018.

[56] J. Wang and G. Joshi, "Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 9709–9758, 2021.

[57] Y. Zhang, M. J. Wainwright, and J. C. Duchi, "Communication-efficient algorithms for statistical optimization," *Advances in neural information processing systems*, vol. 25, p. 3321–3363, 2012.

[58] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," *Advances in neural information processing systems*, vol. 28, p. 1756–1764, 2015.

[59] F. Zhou and G. Cong, "On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization," *arXiv preprint arXiv:1708.01012*, 2017.

[60] S. Shi, X. Chu, and B. Li, "Mg-wfbp: Efficient data communication for distributed synchronous sgd algorithms," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 172–180, IEEE, 2019.

[61] S. Shi, Q. Wang, X. Chu, and B. Li, "A dag model of synchronous stochastic gradient descent in distributed deep learning," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 425–432, IEEE, 2018.

[62] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters," in *USENIX Annual Technical Conference*, vol. 1, pp. 1–2, 2017.

[63] S. H. Hashemi, S. Abdu Jyothi, and R. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 418–430, 2019.

[64] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed dnn training," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 132–145, 2019.

[65] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[66] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 212–229, 2019.

[67] W.-Y. Lee, Y. Lee, J. S. Jeong, G.-I. Yu, J. Y. Kim, H. J. Park, B. Jeon, W. Song, G. Kim, M. Weimer, *et al.*, "Automating system configuration of distributed machine learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2057–2067, IEEE, 2019.

[68] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 949–957, IEEE, 2018.

[69] A. Dutta, E. H. Bergou, A. M. Abdelmoniem, C.-Y. Ho, A. N. Sahu, M. Canini, and P. Kalnis, "On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3817–3824, 2020.

[70] S. Shi, Q. Wang, X. Chu, B. Li, Y. Qin, R. Liu, and X. Zhao, "Communication-efficient distributed deep learning with merged gradient sparsification on gpus," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 406–415, IEEE, 2020.

[71] S. Shi, Z. Tang, Q. Wang, K. Zhao, and X. Chu, "Layer-wise adaptive gradient sparsification for distributed deep learning with convergence guarantees," *arXiv preprint arXiv:1911.08727*, 2019.

[72] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," *Advances in Neural Information Processing Systems*, vol. 31, p. 4452–4463, 2018.

[73] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," *Advances in Neural Information Processing Systems*, vol. 31, p. 5977–5987, 2018.

[74] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[75] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, and X. Chu, "A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2238–2247, IEEE, 2019.

[76] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu, "A convergence analysis of distributed sgd with communication-efficient gradient sparsification," in *IJCAI*, pp. 3411–3417, 2019.

[77] N. Dryden, T. Moon, S. A. Jacobs, and B. Van Essen, "Communication quantization for data-parallel training of deep neural networks," in *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pp. 1–8, IEEE, 2016.

[78] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp: Adaptive residual gradient compression for data-parallel distributed training," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, pp. 2827–2835, AAAI Press, 2018.

[79] S. Shi, X. Chu, K. C. Cheung, and S. See, "Understanding top-k sparsification in distributed deep learning," *arXiv preprint arXiv:1911.08772*, 2019.

[80] A. M Abdelmoniem, A. Elzanaty, M.-S. Alouini, and M. Canini, "An efficient statistical-based gradient compression technique for distributed training systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 297–322, 2021.

[81] J. Fang, H. Fu, G. Yang, and C.-J. Hsieh, "Redsync: reducing synchronization bandwidth for distributed deep learning training system," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 30–39, 2019.

[82] Z. Zhang and C. Wang, "Mipd: An adaptive gradient sparsification framework for distributed dnns training," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3053–3066, 2022.

[83] A. Shanbhag, H. Pirk, and S. Madden, "Efficient top-k query processing on massively parallel hardware," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 1557–1570, 2018.

[84] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[85] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," *arXiv preprint arXiv:1802.06058*, 2018.

[86] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pp. 676–691, 2021.

[87] T. Dettmers, "8-bit approximations for parallelism in deep learning," *arXiv preprint arXiv:1511.04561*, 2015.

[88] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," in *International Conference on Machine Learning*, pp. 3252–3261, PMLR, 2019.

[89] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*, pp. 560–569, PMLR, 2018.

[90] A. T. Suresh, X. Y. Felix, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *International conference on machine learning*, pp. 3329–3337, PMLR, 2017.

[91] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and fault tolerant," *arXiv preprint arXiv:1810.05291*, 2018.

[92] F. Faghri, I. Tabrizian, I. Markov, D. Alistarh, D. M. Roy, and A. Ramezani-Kebrya, "Adaptive gradient quantization for data-parallel sgd," *Advances in neural information processing systems*, vol. 33, pp. 3174–3185, 2020.

[93] S. Horvóth, C.-Y. Ho, L. Horvath, A. N. Sahu, M. Canini, and P. Richtárik, "Natural compression for distributed deep learning," in *Mathematical and Scientific Machine Learning*, pp. 129–141, PMLR, 2022.

[94] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, "Atomo: Communication-efficient learning via atomic sparsification," *Advances in Neural Information Processing Systems*, vol. 31, pp. 9872–9883, 2018.

[95] M. Cho, V. Muthusamy, B. Nemanich, and R. Puri, "Gradzip: Gradient compression using alternating matrix factorization for large-scale deep learning," in *NeurIPS*, 2019.

[96] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Practical low-rank communication compression in decentralized deep learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14171–14181, 2020.

[97] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr, "Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed cnn training," *Advances in Neural Information Processing Systems*, vol. 31, p. 5129–5139, 2018.

[98] A. Sahu, A. Dutta, A. M Abdelmoniem, T. Banerjee, M. Canini, and P. Kalnis, "Rethinking gradient sparsification as total error minimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8133–8146, 2021.

[99] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[100] H. Xu, K. Kostopoulou, A. Dutta, X. Li, A. Ntoulas, and P. Kalnis, "Deepreduce: A sparse-tensor communication framework for federated deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21150–21163, 2021.

[101] H. Shan, S. Williams, and C. W. Johnson, "Improving mpi reduction performance for manycore architectures with openmp and data compression," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pp. 1–11, IEEE, 2018.

[102] G. Yan, S.-L. Huang, T. Lan, and L. Song, "Dq-sgd: Dynamic quantization in sgd for communication-efficient distributed learning," in *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pp. 136–144, IEEE, 2021.

[103] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*, pp. 1737–1746, PMLR, 2015.

[104] C. M. De Sa, C. Zhang, K. Olukotun, and C. Ré, "Taming the wild: A unified analysis of hogwild-style algorithms," *Advances in neural information processing systems*, vol. 28, p. 2674–2682, 2015.

[105] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang, "Stronger generalization bounds for deep nets via a compression approach," in *International Conference on Machine Learning*, pp. 254–263, PMLR, 2018.

[106] Y. Li, T. Ma, and H. Zhang, "Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations," in *Conference On Learning Theory*, pp. 2–47, PMLR, 2018.

[107] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, pp. 3478–3487, PMLR, 2019.

[108] H. Tang, X. Lian, S. Qiu, L. Yuan, C. Zhang, T. Zhang, and J. Liu, "Deepsqueeze: Decentralization meets error-compensated compression," *arXiv preprint arXiv:1907.07346*, 2019.

[109] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations," *Advances in Neural Information Processing Systems*, vol. 32, pp. 14668–14679, 2019.

[110] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proceedings of the 2018*

International Conference on Management of Data, pp. 1269–1284, 2018.
[111] H. Lim, D. G. Andersen, and M. Kaminsky, "3lc: Lightweight and effective traffic compression for distributed machine learning," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 53–64, 2019.
[112] S. Shi, X. Zhou, S. Song, X. Wang, Z. Zhu, X. Huang, X. Jiang, F. Zhou, Z. Guo, L. Xie, *et al.*, "Towards scalable distributed training of deep learning on public cloud clusters," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 401–412, 2021.
[113] A. Albasyoni, M. Safaryan, L. Condat, and P. Richtárik, "Optimal gradient compression for distributed and federated learning," *arXiv preprint arXiv:2010.03246*, 2020.
[114] W.-N. Chen, P. Kairouz, and A. Ozgur, "Breaking the communication-privacy-accuracy trilemma," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3312–3324, 2020.
[115] E. Gorbunov, D. Kovalev, D. Makarenko, and P. Richtárik, "Linearly converging error compensated sgd," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20889–20900, 2020.
[116] M. Safaryan, E. Shulgin, and P. Richtárik, "Uncertainty principle for communication compression in distributed and federated learning and the search for an optimal compressor," *Information and Inference: A Journal of the IMA*, vol. 11, no. 2, pp. 557–580, 2022.
[117] S. Agarwal, H. Wang, K. Lee, S. Venkataraman, and D. Papailiopoulos, "Adaptive gradient communication via critical learning regime identification," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 55–80, 2021.
[118] J. Guo, W. Liu, W. Wang, J. Han, R. Li, Y. Lu, and S. Hu, "Accelerating distributed deep learning by adaptive gradient quantization," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1603–1607, IEEE, 2020.
[119] V. Gupta, D. Choudhary, P. T. P. Tang, X. Wei, X. Wang, Y. Huang, A. Kejariwal, K. Ramchandran, and M. W. Mahoney, "Fast distributed training of deep neural networks: Dynamic communication thresholding for model and data parallelism," *arXiv preprint arXiv:2010.08899*, 2020.
[120] S. U. Stich and S. P. Karimireddy, "The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication," *arXiv preprint arXiv:1909.05350*, 2019.
[121] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik, "Distributed learning with compressed gradient differences," *arXiv preprint arXiv:1901.09269*, 2019.
[122] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.
[123] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.
[124] S.-Y. Zhao, Y.-P. Xie, H. Gao, and W.-J. Li, "Global momentum compression for sparse communication in distributed sgd," *arXiv preprint arXiv:1905.12948*, 2019.
[125] I. Markov, H. Ramezanikebrya, and D. Alistarh, "Cgx: adaptive system support for communication-efficient deep learning," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, pp. 241–254, 2022.
[126] S. Gan, X. Lian, R. Wang, J. Chang, C. Liu, H. Shi, S. Zhang, X. Li, T. Sun, J. Jiang, *et al.*, "Bagua: scaling up distributed learning with system relaxations," *arXiv preprint arXiv:2107.01499*, 2021.
[127] Z. Song, Y. Gu, J. Qi, Z. Wang, and G. Yu, "Ec-graph: A distributed graph neural network system with error-compensated compression," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 648–660, IEEE, 2022.
[128] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.
[129] Nvidia, "Nvidia collective communications library (nccl)." https://developer.nvidia.com/nccl, Feb 2023. [Online].

Zhangqiang Ming is currently working toward the Ph.D. degree in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He received his M.S. degree in the College of Computer Science, Sichuan University, Chengdu, China, in 2022. His research interests include high-performance networks and architecture, distributed machine learning/deep learning systems, and distributed computing.

Wenxiang Zhou is currently working toward the master's degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He is interested in distributed systems and operating systems.

Xinjue Zheng is currently working toward the master's degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests include distributed systems and distributed computing.