Journal of Systems Architecture 142 (2023) 102927

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Communication compression techniques in distributed deep learning: A survey☆

Zeqin Wang [a], Ming Wen [a], Yuedong Xu [a,*], Yipeng Zhou [b], Jessie Hui Wang [c], Liang Zhang [d]

[a] School of Information Science and Technology, Fudan University, Shanghai, 200433, China
[b] FSE, School of Computing, Macquarie University, Macquarie Park, NSW 2113, Australia
[c] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, 100084, China
[d] Huawei Technologies, Nanjing Research Center, Nanjing, 210096, China

ARTICLE INFO

Keywords:
Distributed deep learning
Communication compression
Sparsification
Quantization

ABSTRACT

Nowadays, the training data and neural network models are getting increasingly large. The training time of deep learning will become unbearably long on a single machine. To reduce the computation and storage burdens, distributed deep learning has been put forward to collaboratively train a large neural network model with multiple computing nodes in parallel. The unbalanced development of computation and communication capabilities has led to training time being dominated by communication time, making the communication overhead a major challenge toward efficient distributed deep learning. Communication compression is an effective method to alleviate communication overhead, and it has evolved from simple random sparsification or quantization to versatile strategies or data structures. In this survey, existing communication compression techniques are reviewed and classified to provide a bird's eye view. The main properties of each class of compression methods are analyzed, and their applications or theoretical convergence are described if necessary. This survey is potentially helpful for researchers and engineers to understand the up-to-date achievements on the communication compression techniques that accelerate the training of large deep learning models.

1. Introduction

Deep learning is an important branch of machine learning that uses multiple neural processing layers with complex structure or consisting of multiple nonlinear transformations to extract high-level features of data [1]. With the rapid development of big data and computing power, deep learning has witnessed a tremendous growth of applications in image processing [2], natural language processing [3], speech recognition [4], data mining [5] etc., significantly outperforming domain specific algorithms and classical machine learning methods. A recent surge of interest is employing deep learning to address science problems in chemistry, biology, physics and mathematics [6].

Today's deep learning models are usually gigantic in terms of their large number of neural network layers and huge amount of model parameters, combined with the huge volume of training data. For instance, BERT-base and BERT-large models contain more than 110 million and 340 million parameters respectively [7], and GPT-4 even owns 175 billion parameters. The ImageNet dataset contains 14 million images for image recognition [8] and WMT-2014 contains 36 million bilingual pairs for natural language processing [9]. A single machine is nearly impossible to load all the training data or the model to its local memory. The computing power of a machine also hampers the speed of training large deep learning models. With stochastic gradient descent (SGD) method as the training algorithm, computing the gradients by traversing all the data samples in one epoch is time-consuming. In addition, the training data are dispersed in multiple sites that do not allow the exchange or aggregation of raw data due to privacy concerns. Therefore, an inevitable trend is to perform learning calculations in parallel on distributed computing clusters for large deep neural networks (DNNs).

Distributed DNN training splits the centralized computation into multiple parallel computations. Data parallelism is the most common distributed mode. Each machine has access to a specific partition of the data set. A typical distributed training architecture is shown in Fig. 1 that consists of a central server (namely parameter server) and

☆ This work was supported in part by the Natural Science Foundation of China under Grant Grant 62072117, the Shanghai Natural Science Foundation under the Grant 22ZR1407000 and Huawei collaborative project on stochastic network modeling.

* Corresponding author.
E-mail addresses: zeqinwang21@m.fudan.edu.cn (Z. Wang), mwen19@fudan.edu.cn (M. Wen), ydxu@fudan.edu.cn (Y. Xu), yipeng.zhou@mq.edu.au (Y. Zhou), jessiewang@tsinghua.edu.cn (J.H. Wang), zhangliang1@huawei.com (L. Zhang).

https://doi.org/10.1016/j.sysarc.2023.102927
Received 4 March 2023; Received in revised form 18 May 2023; Accepted 17 June 2023
Available online 22 June 2023

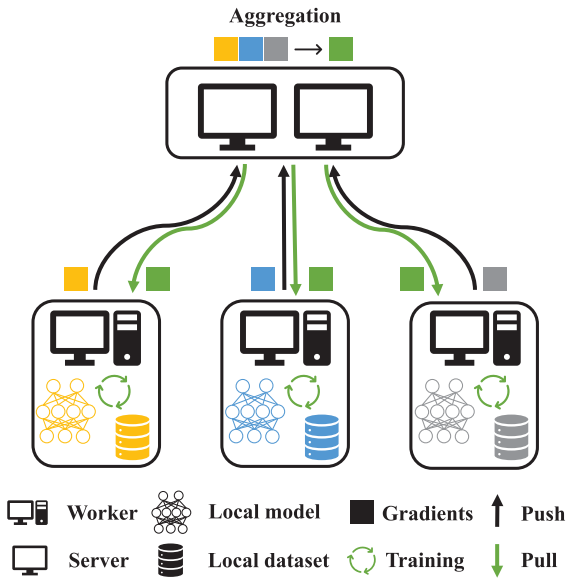


Fig. 1. Distributed deep learning.

multiple local workers. Each worker computes the local gradient based on its data samples and pushes it to the server for aggregation. After the central server generates a global model from received local gradients, all the workers pull this new model and start the next training round. This procedure repeats many rounds until the DNN model converges. The distributed training relieves the burden of centralized computation, while eliciting a huge communication overhead between the parameter server and the local workers. When the DNN model is large and the bandwidth is limited, the communication time is comparable to the computation time, and even overwhelms the latter, thus undermining the benefit of distributed learning.

Communication reduction is one of the primary challenges in improving the efficiency of distributed deep learning. Usually, two direct approaches are adopted. First, each worker chooses a batch of data samples at an iteration in the SGD based training. The large batch size significantly reduces the number of iterations needed to reach the certain objective, thus diminishing the communication load. Second, each worker periodically pushes its gradient to the parameter server after performing multiple local iterations instead of every iteration.

Despite their advantages in communication reduction, a large batch size usually causes the DNN training to converge to an unsatisfactory local minimum, and a long update interval might deviate the local models from the global model that slows down the convergence rate. An intriguing property of distributed DNN training is that it is *tolerable* to the imperfection of received local models. On one hand, most of the gradients in a local model are extremely small, and their influence on the convergence rate is very likely marginal. On the other hand, if the fewer number of bits is used to represent each parameter, the performance of the DNN model and the convergence rate measured in the training rounds might not be affected under certain situations. These observations give rise to a cluster of communication reduction techniques, namely, "communication compression". Based on its core idea, communication compression techniques are roughly categorized into two groups, the *sparsification* and the *quantization* of gradients to be transferred.

Sparsification methods only transmit the *important* gradients that govern the convergence rate of DNN training in each round of parameter synchronization. In general, when gradients are used as the content of communication, the communication sparsification method selects part of these elements for transmission, and the gradient elements that are not selected are replaced with zero values during the decompression process. Quantization methods involve discretizing continuous gradient values and mapping them to specified values within a low precision range. These specified values are represented using fewer bits than the original values, which reduces the size of each gradient instead of the number of gradients. During transmission, the quantized values can be represented with fewer bits, resulting in a decreased communication overhead and shorter gradient synchronization time.

1.1. Motivations

As the most effective way of alleviating communication burdens in distributed deep learning, the compression of local models has been extensively studied in recent years. A few comprehensive surveys [10–12] convey the recent progresses on communication-efficient DNN training, yet their coverage is too broad to examine the communication compression in details. An interesting survey [13] is also dedicated to communication compression that carefully describes the important compression algorithms. However, it only classifies the existing algorithms at the algorithmic perspective. The design rationales of compression approaches adapted to various applications or environments are not included. In addition, the comparison of different compression approaches in their convergence rates is not emphasized in the previous surveys.

Our survey focuses on communication compression in the distributed DNN training with data parallelism. This survey collects the current mainstream communication compression techniques and describes them in detail. We present a taxonomy of the state-of-the-art that consists of four types of studies: sparsification, quantization, hybrid compression and context specific compression. We discuss various approaches for sparsification, including relative value-based, absolute value-based, and matrix decomposition-based selection methods. For quantization, we cover truncated expression and code-book mapping-based selection methods. The hybrid approaches refer to combination of sparsification and quantization. The context specific approaches integrate application scenarios and general compression frameworks. We further explore the differences of these methods considering factors such as the layer-wise versus global compression, unidirectional versus bidirectional transmission, i.i.d versus non-i.i.d data distribution, extra computational overhead, and compare the theoretic convergence rate and training performance of the selected algorithms. Our survey carefully examines a number of new algorithms, and incorporate a set of new scenarios. In addition, we move one step further to compare the advantages and disadvantages of versatile compression techniques.

1.2. Paper organization

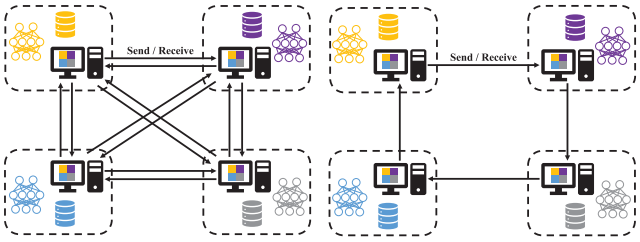
The remainder of the survey is organized as follows. Section 2 offers a brief introduction to communication architecture, communication reduction strategies and communication compression. Section 3 provides a taxonomy of communication compression. Sections 4, 5, 6 and 7 review the compression methods in detail. The comparison of different methods is included in Section 8. We conclude the survey in Section 9.

2. Preliminary

In this section, we introduce the communication architecture of distributed deep learning. The classical communication reduction techniques are briefly described as the appetizer.

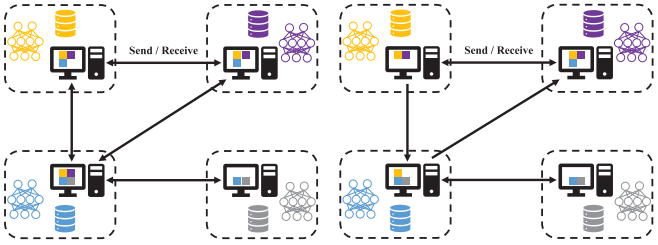
2.1. Distributed deep learning

Distributed deep learning is commonly classified as the Parameter Server (PS) architecture, the All-Reduce architecture and the decentralized architecture. They mainly differ in the ways that computing nodes are interconnected and in the approaches that the global consensual model is reached. As a consequence, the design rationales in these distributed learning architectures are different.



(a) All-Reduce (b) Ring All-Reduce

Fig. 2. All-Reduce and Ring All-Reduce.



(a) Undirected communication (b) Directed communication

Fig. 3. Communication based on a graph topology.

In the PS architecture shown in Fig. 1, the computing nodes are tagged with two roles, the *worker* and the *PS*. Each worker uses a subset of training data to generate his local DNN model. The parameter server is responsible for collecting all the local gradients from the workers and performing the aggregation. After the aggregated gradient is generated, each worker pulls it back for model update and starts the next-round model training. This procedure repeats many rounds until the global model converges. The PS architecture is throttled by the communication bottleneck connecting the server and the workers. The uplink bandwidth is shared by all the workers, so is the downlink. In light of gigantic sizes of DNN models, the communication time is commensurate to the local computing time, and may even overwhelm the latter so that the advantage of distributed training vanishes. As the computing capability of hardware (e.g. GPU, FPGA and ASIC) improves, the communication efficiency of the PS architecture degrades accordingly. In addition, when the workers are geographically distributed and connected through public Internet links, the limited bandwidth prolongs the wall-clock training time. To alleviate the communication burden, distributed parameter servers are adopted so that each PS only processes a fraction of the parameters or gradients, and all the parameter servers jointly achieves the goal of global aggregation.

All-Reduce is an operation that reduces the target arrays in all processes to a single array and returns the resultant array to all processes. In distributed deep learning, the All-Reduce architecture shown in Fig. 2(a) enforces the gradient aggregation and the parameter synchronization at every worker in the absence of a central server. It implements the communication process of gradients in versatile forms, in which the Ring All-Reduce is the most notable one. The Ring All-Reduce architecture resembles a directed ring topology as shown in Fig. 2(b). The local gradient at a worker is sliced into multiple segments, with each slot allowing only one segment to be delivered from a worker to his neighbor. This Ring All-Reduce procedure consists of the "Scatter-Reduce" and the "All-Gather" stages. In the first stage, each worker aggregates the received segment with his own and relays it to his neighbor step by step until every worker obtains a complete aggregated segment. In the second stage, each worker sends the fully aggregated segment to his neighbor via multiple rounds so that the global consensual model is reached. The All-Reduce architecture has the potential of avoiding single-link bottleneck, while coordinating the sequential transmissions is cumbersome especially when the number of workers is large.

Both the PS and the All-Reduce architectures require each worker to obtain the global model at every iteration. Their major difference lies in whether the aggregation is performed at the parameter server or at the distributed workers. In practice, a worker may communicate only with its neighbors, thus forming a general decentralized topology. Instead of obtaining a global model, each worker aggregates the local modes from its neighbors to yield a partial model. The partial aggregation between two workers can be bidirectional as shown in Fig. 3(a) or unidirectional as shown in Fig. 3(b). Compared with PS and All-Reduce, the decentralized learning greatly reduces the communication load since the global model does not go through every worker. In exchange, the lack of the global consensus prolongs the number of training rounds to reach the predetermined accuracy.

2.2. Communication reduction

Reducing the number of communication rounds is an effective approach to cut off the communication overhead in distributed deep learning. As the basic ideas, one can choose a large batch size to accelerate the convergence rate, or to employ the periodical communication after multiple rounds of local computations.

2.2.1. Large batch training

Batch size is an important hyperparameter in the training of DNN models that refers to the number of fed data samples at an iteration. A large batch size yields a more accurate estimation of the gradient on the entire dataset, thus potentially reducing the number of training rounds toward convergence. Fewer training rounds mean fewer rounds of model update. There are drawbacks to choosing a large batch size. Because DNN models are usually non-convex, the large-batch training is more inclined to being trapped at a sharp local minimum with large surrounding curvatures [14]. When the gaps between the local minima and the global minimum are large, the performance of DNN models will be hardly satisfactory. Many studies [14–16] have confirmed that the large batch training reduces the model generalizability. Ma et al. [17] experimentally showed that when the batch size exceeds a certain threshold, the number of iterations required for the model to eventually converge increases on the contrary.

2.2.2. Periodic communication

Periodic communication is a commonly used method to reduce the training traffic, in which multiple local iterations are performed at a node before gradient synchronization. The reduction of communication is proportional to the number of local iterations. The costs of periodic communication are the prolonged training rounds and the loss of final accuracy. When the number of local iterations increases, the local model may deviate from the global model considerably, so that the convergence becomes slower in terms of the number of training rounds to reach a certain level of accuracy. More severely, too many local updates also drive the model toward the local optimum, thus impairing the model accuracy and its generalizability [18]. Such a delicate balance should be carefully treated through the tuning of the number of local iterations.

2.3. Communication compression

Beyond large-batch training and periodic communication, there exists an intriguing method, namely *communication compression*, that reduces the amount of gradients to be transmitted or encodes a gradient

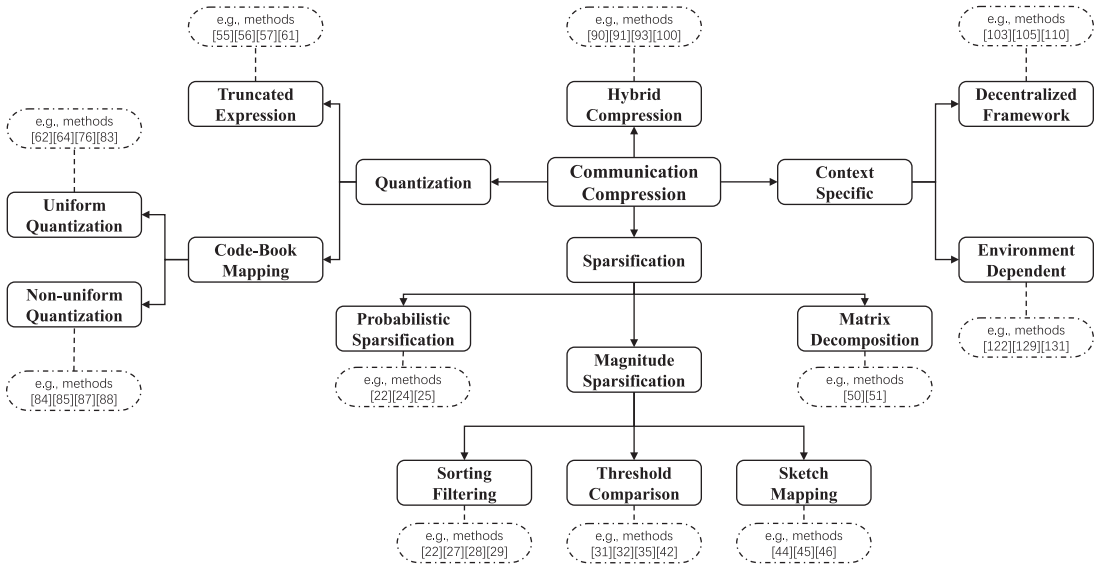


Fig. 4. The hierarchical taxonomy of communication compression techniques.

with less bits at each round. The transmission time of a compressed model is then shorter. The underlying reason of communication compression lies in that the distributed DNN training is robust to imprecise gradients. Consider communication compression in the parameter server architecture as a use case. When the received local models are imprecise, the aggregated model may deviate from the true global model, thus leading to the increased number of iterations to reach the predefined accuracy. However, the wall-clock time of each iteration is significantly reduced so that the total training time decreases considerably.

The disadvantage of communication compression is the imprecision of the local and resultant global models, whereas such imprecision can be mitigated through compensation techniques. The error accumulation compensation stores the current compression error locally and compensates for it in subsequent iterations. It ensures that information is not lost, but only delayed. The earliest error accumulation compensation method is inspired by Sigma-Delta modulation [19], where the errors between the genuine and the compressed gradients are cached locally. These errors are added to the local gradients computed at the next iteration, and the refined local gradients are compressed subsequently before transmission. After multiple iterations, the small gradients will be included in the model training so as to calibrate the deviation of the global model with communication compression. The gradient compression as well as the error compensation techniques are independent of communication architectures, i.e. deployable in both parameter server and All-Reduce, and scalable to versatile learning algorithms. New error accumulation compensation methods have been proposed currently [20,21].

The simple error compensation technique is not the optimal choice for variance-reduction optimization algorithms (e.g., Momentum, Stochastic Recursive Momentum and Implicit Gradient Transport, etc.). In fact, its implementation may potentially hinder the convergence speed of Federeated Learning [20]. **ErrorCompensatedX** [20] (ECX) uses the compression error of the first two iterations for compensation. It deployed gradient compression on both the server and the worker side in the PS architecture. ErrorCompensatedX borrows the idea of a low-pass filter. The compensation is performed after the compression error from the previous two iterations is adjusted through a local scaling factor and the low-pass filter.

3. Overview of compression taxonomy

According to the purpose of compression, we can classify the communication compression techniques into four major categories including sparsification, quantization, hybrid compression and context specific compression in Fig. 4. The former three categories focus on different ways to squeeze the size of gradients. Sparsification reduces the number of transmitted elements. Quantization decreases the bit-width of each element. Hybrid compression reduces both the number of transmitted elements and the bit-width of each element. On the other hand, context specific compression takes a different approach, as it prioritizes exploiting the efficiency of the compression techniques within a given environment rather than reducing communication volume. This makes context specific compression applicable to most compression algorithms.

To achieve effective communication sparsification, the selection method of gradient elements plays a crucial role. This selection determines the sparsity level of the gradients that contributes differently to model updates. According to the selection strategies, communication sparsification can be further classified into three categories: probabilistic sparsification, magnitude sparsification and matrix decomposition sparsification. In probabilistic sparsification, each element has the opportunity to be selected for communication. This technique involves assigning a series of identical or different probabilities to each element, and then selecting the elements randomly based on their assigned probabilities. Magnitude sparsification uses a hard metric that specifies the criteria for elements that can be transmitted. It selects the gradient elements with top large magnitude for communication based on the observation that gradient elements with larger absolute values are more important to the model update. To be more specific, there are several ways to filter these elements, each with varying levels of complexity and operational difficulty. The first method is to use a sorting algorithm. The second method involves comparing each element with a threshold value to determine if it should be transmitted. The third method involves using a sketch data structure to identify the top elements. The selection is carried out with a holistic perspective in matrix decomposition sparsification. The original large matrix is approximately decomposed into several matrices of small size. These small matrices are transmitted among workers.

The essence of communication quantization is to reduce the bit-width of each gradient element. Two main techniques used to achieve this goal are truncated expression and code-book mapping. Truncated expression directly reduces the precision of the element from full precision to lower precision or even a sign representation. This is mainly achieved by encoding the elements with fewer bits. Code-book mapping builds up a code-book and maps continuous element values onto the discrete quantization values. Only the bit representation of

the quantization level is transmitted. The focus of code-book mapping is the construction of the code-book, especially the setting of the quantization values. It determines whether the mapping result can capture the characteristics of the element distribution. According to the interval between different quantization values, code-book mapping can be further subdivided into uniform quantization and non-uniform quantization.

Considering that the underlying mechanisms of sparsification and quantization are orthogonal, hybrid compression methods come up with a combined version. In general, the technique quantifies the gradient elements after sparsification. Sometimes, it compresses the quantized elements by sparse encoding due to the sparsity of elements after quantization.

Context specific compression refers to the adaptation of communication compression in various application scenarios. Communication architecture and resources are the two factors that have the greatest impact on compression technology in practical applications. We can divide context specific compression into two categories according to these two factors. One category is decentralized compression frameworks. Decentralized learning has a different communication pattern where parameters are transmitted between connected neighbors instead of aggregated by the parameter server. This may incur additional compression error which will cause learning divergence. Therefore, decentralized communication compression design has its own unique characteristics to alleviate this issue. Another category is the environment dependent compression framework. In the real-world, the computation and communication resources are not balanced among workers and the network environment is always dynamic. Environment dependent compression frameworks are specially designed to improve the efficiency of compression methods according to the variability or imbalance of resources.

4. Communication sparsification

4.1. Communication sparsification basics

The key idea of communication sparsification is to transmit only large gradient elements that have great impact on the convergence of DNN model training. At each iteration, a worker typically selects large elements from the gradient tensor, and transmits two condensed vectors to the parameter server. One vector contains the values of the selected gradient elements, and the other corresponds to their indices. An illustration of the sparsified gradient is shown in Fig. 5. The index vector can be deemed as the cost of gradient compression because it is hard to be further compressed. After the parameter server receives the compressed gradient vector, it is reconstructed into a complete tensor. The element values whose indices do not appear in the index vector are filled with zeros.

4.2. Probabilistic sparsification

Probabilistic sparsification filters out the gradient elements in the communication set according to certain random criteria. The probability of each element being selected can be identical, or different depending on its value for the purpose of faster convergence.

Random-k [22] randomly selects k gradient elements out of a total of n elements. It possesses two important properties, i.e. unbiasedness and low operational overhead. The expectation of the sparsified gradient after tensor reconstruction is exactly the real gradient. Meanwhile, Random-k does not involve the sorting of gradient elements, which yields very light computational complexity. **Block-Random-k** [23] selects the first gradient element randomly and then the subsequent $(k-1)$ elements immediately. Compared with Random-k, Block-Random-k further reduces the compression overhead by performing only one random data access.

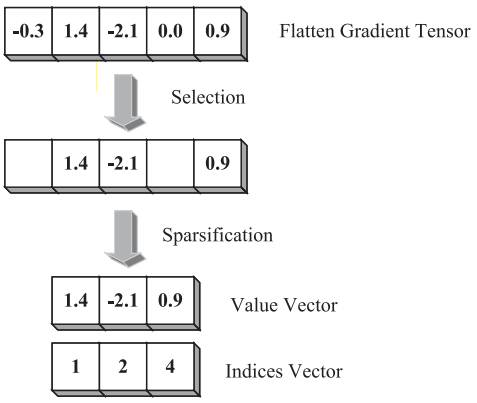


Fig. 5. Gradient sparsification.

Random-k ensures the unbiasedness of the gradient while magnifying its variance. In order to strike the balance between sparsity and variance, **GradSparse** [24] introduces a new probabilistic sparsification method. Given a fixed variance budget, it builds a convex optimization problem to decide the probability of selecting each element in a gradient vector. One important finding of GradSparse is that the top elements (i.e. top-k) cannot be dropped. The probability of choosing each element is given by

$$p_i = \begin{cases} 1, & i \in S_{topk} \\ (\epsilon \sum_{j=1}^{d} g_{(j)}^2 + \sum_{j=k+1}^{d} g_{(j)}^2)^{-1} |g_i| (\sum_{j=k+1}^{d} |g_{(j)}|), & i \notin S_{topk}. \end{cases}$$

Here, g refers to the original gradient with d elements and $g_{(j)}$ is the jth gradient element after sorting. S_{topk} is the set of coordinates with top-k gradient magnitudes. ϵ is the variance budget. The probability is approximated by a greedy algorithm to mitigate the computational overhead of sorting.

Probcomp-LPAC [25] also randomly selects gradient elements for transmission without sorting them. It first computes the maximum absolute value in a gradient tensor. Then, the probability of selecting an element depends on the difference between these two values,

$$p_i = \frac{|g_i|}{\max(|\max(g)|, |\min(g)|)}.$$

The smaller the difference, the higher the selection probability. As an advantage, Probcomp-LPAC has light computational overhead. In addition, Probcomp-LPAC sets different compression ratios on different neural network layers. The compression ratio c_l of layer l is given by

$$c_l = [r + (p \times index^3 + q \times index^2 + d \times index)] \times c.$$

Here, $r \in [0.1]$ is the initial change rate, and p, q and d are hyperparameters to control the growth of the change rate. The symbol $index$ is marked as the descending order of layer l after being sorted according to the layer size. With the layer-size dependent compression ratios, Probcomp-LPAC compresses the layers with more parameters aggressively and those with less parameters conservatively.

4.3. Magnitude sparsification

Discarding gradient elements with smaller absolute values has less impact on the convergence rate than discarding larger ones [26]. The reason is simple, i.e. the gradient elements with larger absolute values contribute more in the decent direction toward the optimum. A natural way of communication compression is to filter out the gradient elements based on their magnitudes. We hereby describe two set of approaches depending on whether the gradient elements are sorted or not.

4.3.1. Sorting and filtering

Top-k [22] selects k elements with the largest absolute values as the communication set via a sorting algorithm. The error accumulation compensation is adopted to combat the biased compression of Top-k. The unselected small elements are not discarded, but delayed for transmission in future iterations. Top-k with error compensation is akin to asynchronous SGD in terms of the delayed transmission, while in fact they differ critically. In Top-k with error compensation, only a fraction of gradient elements is delayed and the maximum delay can be infinite. In asynchronous SGD, all the elements among the same gradient have the same delay, and the delay is usually assumed to be bounded. One potential drawback of Top-k is the high compression overhead, i.e. the time complexity of sorting is nontrivial when the communication bandwidth is high.

Top-k sparsification gives rise to many extensions. Barnes et al. presented **rTop-k** [27], a combination of Top-k and Random-k, which first selects the largest r ($r > k$) elements, and then randomly selects k elements in these elements as the communication set for transmission. Note that the design of rTop-k is rooted in the distributed statistical estimation theory. A sparse Bernoulli distribution is used to characterize the distribution of gradient values. The larger values are observed as '1' and the smaller values are observed as '0'. For the lower bound on estimation error under sparse Bernoulli distribution, this work compares the result of random subsampling with that of arbitrary estimators. It shows that rTop-k is optimal in terms of order up to logarithmic factors.

Both vanilla Top-k and rTop-k transmit the sparsified gradient to the parameter server, while leaving the downloading process of the aggregated model unaltered. In **gTop-k** [28], not only the workers execute the Top-k sparsification scheme, but also the parameter server uses it over the aggregated gradient and sends back the sparsified one. The advantage of gTop-k is the significant reduction of downlink traffic load. Because the indices of the top-k elements computed by each worker vary greatly, after averaging the number of gradient elements scales up by a factor of the number of workers. With the Top-k sparsification at the parameter server, the downlink bottleneck is mitigated.

Scalecom [29] uses the local gradient similarity of different workers to perform a "global" Top-k algorithm. When the number of workers becomes large, two problems arise. One is that the downlink communication becomes the bottleneck, the other is the scaled learning rate required for large-scale training increases the gradient noise significantly. To reduce the downlink traffic, Scalecom enforces all the workers to select the gradient elements of the same indices. It introduced the Cyclic Local Top-k (CLT-k) algorithm so that the indices of the global model coincide with those of one leading worker. As an additional advantage, Scalecom is adaptable to the pipelined transmission of the Ring All-Reduce architecture. To tackle the second problem, ScaleCom designed a low-pass filter that reduces the noise amplification caused by the scaled learning rate, and increases the gradient similarity across different workers.

LGC [30] asserts that the gradients between workers are correlated, and it exploits the redundancy between workers to improve the compression efficiency. LGC divides layer gradients of each worker into the public component and the innovation component. The public component shared by all the workers can be eliminated and only the innovation component unique to each worker needs to be captured. The innovation component is the top-magnitude gradients.

LGC designs the corresponding autoencoders realized with lightweight neural networks for the Parameter Server architecture and the Ring All-Reduce architecture, respectively. In the Parameter Server architecture with K workers, it requires one encoder and K decoders. The elements from all the workers are sequentially passed to the encoder for encoding with the goal of minimizing the similarity loss. The corresponding decoder reconstructs the gradients with the objective of minimizing reconstruction loss based on the public and innovation component. In the Ring All-Reduce architecture, only one encoder and one decoder are required. The elements in the innovation component are in the same position for each worker. And the rest of the operations are similar to the operations in Parameter Server architecture.

4.3.2. Threshold comparison

An alternative sparsification approach is to use a threshold value to filter out the unimportant gradient elements, thus avoiding the cumbersome sorting operation. In contrast to the sorting-based algorithms, the threshold-based ones need to select an appropriate threshold value, and lose the exact control of the compression ratio.

Threshold-v [31] uses two thresholds, a positive one and a negative one to filter out the gradient elements. Only those above the positive threshold and lower than the negative threshold are selected as the communication set. One clear benefit of Threshold-v is the low computational overhead, and its drawback is the inflexible choice of the thresholds. Deep neural networks differ in their network structures and optimizers so that their optimal thresholds are not coincident.

DGC [32] is a threshold based sparsification approach, although it implements the Top-k selection. At the first step, DGC randomly samples a small fraction of gradient elements, e.g. 0.1% to 1% of the total. The sparsification threshold is obtained by applying a Top-k algorithm on this sampled gradient. This threshold is further used to select large elements in the original gradient. The advantage of DGC is its reduction on the computational complexity of Top-k significantly. If the number of elements filtered out is much higher than expected, an exact threshold is then calculated from the filtered elements. To compensate for the loss of accuracy and gradient staleness due to compression, auxiliary techniques are incorporated such as error accumulation compensation, momentum correction, local gradient clipping, momentum factor masking and warm-up training.

For the momentum SGD, the current DGC with error feedback is not directly applicable. Sparse updates will omit the discounting factor term in model update, changing the optimization direction [33]. In order to correct the direction, momentum correction accumulates the new gradient with momentum instead of the actual gradient. To avoid gradient explosion, local gradient clipping is performed. This method rescales the gradients when the sum of their L2-norms exceeds a certain threshold. Momentum factor masking aims to prevent stale momentum from deflecting optimization. The momentum factors associated with the unsent cumulative gradients are set to zero. In the early stage of training, sparse updates can lengthen the period of neural network changes dramatically. Warm-up training uses a small learning rate and a less aggressive sparsification at the start of training to reduce the impact of early gradients.

GradDrop [34] approximates the threshold by sampling estimation, ensuring that there are enough elements larger than this threshold. It designs the layer-wise thresholds or global threshold for the entire model. Since the difference in gradients across layers can be large, the calculation of global threshold needs to be combined with layer normalization. Without layer normalization, the global threshold may degrade model convergence.

RedSync [35] proposes two efficient implementations of Top-k on GPUs via threshold comparison, where one is the trimmed top-k selection and the other is the threshold binary search selection. In the first method, the initial layer-wise threshold is calculated based on the ratio, the mean and the maximum values of each tensor layer, i.e.,

$$threshold = mean + ratio \times (max - mean).$$

If the number of elements filtered by the threshold is smaller than the expected number, it dynamically decreases the threshold so that more elements can be filtered. The RadixSelect algorithm is performed on the remaining elements of a relatively smaller subset to obtain the accurate sparsified result. The second method searches for the threshold between the first k and $2k$ largest elements (in terms of their magnitudes) via

the canonical binary search. The compression time of RedSync is non-negligible when the size of a layer is large. To reduce the computational overhead, RedSync treats different layer sizes dissimilarly. For a small layer, the top-k method is used directly; for a medium sized layer, the trimmed top-k method is used and for a large layer, the threshold binary search selection is adopted.

MSTop-k [36] considers communication sparsification in a large cluster with multiple machines and multiple cards. This method aims to address the inefficiency of top-k selection on GPUs and the expensive communication overhead of All-Gather. It implements the top-k selection by multiple samplings. MSTop-k first uses the mean value as the first threshold, and then halve or double it as the second threshold. After that, the process is repeated continuously and these two thresholds are updated until the required number of searches is reached. Because the commonly used sorting algorithm involves many irregular memory access, MSTop-k is more efficient on memory care units such as GPUs.

To reduce the communication overhead of All-Gather in large systems, MSTop-k further proposes the hierarchical top-k communication algorithm HiTopKComm. Each worker performs an intra-node Reduce-Scatter operation in parallel so that every GPU has a portion of the local gradients. The MSTop-k selection is used in each GPU to compress the tensor. The corresponding GPU performs All-Gather operation between workers. An intra-node All-Gather is conducted to complete the aggregation. HiTopKComm makes full use of the bandwidth resources of intra-node and inter-node connections to improve communication.

EGC [37] dynamically determines the sparsification threshold of each layer based on the entropy of its gradient elements. The intuition behind this design lies in that a large entropy is inclined to carrying more important information so that less gradient elements should be pruned. At each layer, EGC partitions the range of gradient elements equally into N intervals, and the number of gradient elements fell in each interval is counted subsequently. Let p_i be the fraction of the gradient elements in the ith interval, and let H be the entropy of this layer that has

$$H = -\sum_{i=1}^{N} p_i log p_i.$$

The fraction of gradients sent by the worker is set to $\min\{\frac{H}{K}, 1\}$ where K is the predefined constant. Hence, when H is too small, it is reasonable to send less gradient elements, and send more otherwise. The sparsification threshold can be determined using Quickselect [38].

To compensate for the accuracy loss, EGC incorporates automatic adjustment of the learning rate that adopts to the change of the training loss. EGC can be implemented in different distributed architecture. It proposes the EGC-PS for the parameter server architecture, the EGC-DSGD for the All-Reduce architecture and the EGC-FL algorithm for federated learning.

To compensate for the gradient staleness from error accumulation, **GradSA** [39] performs sparsification based on historical gradient information. It uses the result of Taylor expansion of fresh gradients to replace the original historical gradients and adds a decay factor to the accumulation. In addition, GradSA also adopts a layer-by-layer sparsification approach in order to improve the quality of the sparsified gradients.

Knowing that the standard deviation of gradient elements indicates their degree of dispersion, **SDAGC** [40] leverages it to dynamically calculate the sparsification threshold. For a particular layer l, its threshold is given by

$$threshold_l = \alpha \times \sigma_l,$$

where α is the hyperparameter to control gradient sparsity and σ_l is the standard deviation of gradients. The auxiliary approaches including residual gradient accumulation, local gradient clipping, adaptive learning rate correction and momentum compensation are used for such an aggressive sparsification to avoid the decrease of training accuracy.

Gaussian-K [41] models the gradient distribution as a normal distribution to obtain the threshold estimate. It uses the mean value and variance of the original gradient to construct a normal distribution. The percentage point function is used to calculate the threshold. Since the true gradient distribution does not exactly match the normal distribution, the number of selected elements may deviate from the desired one. To select the desired number of gradients, Gaussian-K adjusts the threshold by shifting it left or right.

SIDCo [42] fits the gradient distribution and estimates the threshold corresponding to the compression ratio based on the fitting results. It proposes three different distributions, which are double exponential distribution, double gamma distribution and double generalized Pareto distribution. Although they fit the true gradient distribution in the DNN well, there are some deviations in the tail of the CDF. To address inaccurate estimation due to the deviations, the single-stage and multi-stage compressions are proposed. For general compression ratios, SIDCo performs only one fitting to obtain an accurate threshold. For very aggressive compression ratios, SIDCo turns them into multiple compression, and uses different distribution models to fit and filter the gradients.

4.3.3. Sketch mapping

The Sketch data structure uses multiple hash functions to map the original elements to a contiguous memory space [43]. It supports two operations, insert and query. Insert operation updates elements in the sketch and query operation estimates the original elements from the sketch. Sketch uses limited spatial resources to estimate the overall information and can achieve a better balance between estimation accuracy and memory usage. As a result, many works make the use of Sketch for communication compression.

Of many Sketch data structures available, Count Sketch is the most commonly used in communication compression. It compresses a vector $g(d)$ into a Sketch $S(g)$ of size $O(\log d)$. On one hand, Count Sketch can obtain the heavy hitters in the original elements and use them as an accurate estimate of the top-k largest elements. Thus, sketch mapping is considered a special kind of sparsification. On the other hand, the linearity of Count Sketch, i.e. $S(g_1)+S(g_2) = S(g_1+g_2)$, makes it widely adopted in distributed training. A major difference between Sketch and general sparsification is that the aggregated result does not scale with the number of workers. As a result, sketch mapping-based sparsification methods do not congest the downlink communication in large-scale distributed systems.

Spring et al. compress auxiliary variables in the optimization algorithm based on Count Sketch [44]. It aims to free up the memory for either a more expressive model or a larger batch training. They empirically find that the auxiliary variables show a power-law distribution during the training. Only a fraction of the elements makes a significant contribution to the model update. In addition, these auxiliary variables are updated in a linear fashion, which is suitable for Sketch. Therefore, the Count Sketch can be used for the compression of these auxiliary variables.

Spring et al. only compress for auxiliary variables and do not endeavor to reduce the communication overhead. **Sketched-SGD** [45] uses Count Sketch to compress gradients in communication and updates the model with global top-k gradients. It requires two rounds of communication between the workers and the parameter server. In the first round, each worker transmits the sketch results of the local gradients. The parameter server aggregates these sketches and recovers the indices of the global top-k elements through the Heavy Mix algorithm. In the second round, the parameter server collects the corresponding elements from all workers for gradient aggregation. To improve the convergence speed, Sketched-SGD also employs gradient error accumulation compensation, momentum correction and momentum factor masking techniques similar to DGC.

FetchSGD [46] makes use of Count Sketch for communication compression in Federated Learning. In Federated Learning, there are

problems with client drop, in addition to inefficient communication. Sketched-SGD requires two rounds of communication to complete an iteration. However, a client may participate only once during all of training. The momentum and error information on the clients is easily lost. Therefore, FetchSGD recovers the top-k elements directly with the merge of sketches and completes the momentum and error accumulation compensation on the aggregator.

4.4. Matrix decomposition sparsification

The gradient elements of a DNN layer can be viewed as a matrix, where the number of columns corresponds to the input neurons and the number of rows corresponds to the output neurons. Using singular value decomposition (SVD), a matrix can be expressed as a product of three matrices where the singular value matrix is in the middle. If we retain a subset of large singular values, the corresponding columns of the first matrix as well as the corresponding rows of the third matrix, the original matrix can be constructed approximately. Therefore, instead of transmitting the full gradient tensors, matrix decomposition only sends the selected values used for gradient reconstruction, thus greatly reducing the traffic volume. Similar to above mentioned sparsification approaches, matrix decomposition transmits less elements in the important basis vectors.

The advantages and disadvantages of matrix decomposition sparsification are very intuitive. First, in contrast to ordinary sparsification methods, it does not designate the indices of selected elements. Therefore, it is naturally integrated in the All-Reduce architecture without any compression coordination. Second, matrix decomposition sparsification does not require the carefully designed error compensation techniques to ensure model convergence [47]. Third, the use of low-rank update can be treated as the spectral regularization [48], which facilitates the model generalizability [49]. However, this matrix decomposition has some criticisms that the computational overhead is huge and the compression ratio can be set arbitrarily.

GradiVeQ [50] compresses the gradient matrix g with the whitening vector μ_m and the matrix $U_{d,m}$. μ_m is the average result of the gradient matrix in the previous L_t uncompressed iterations. $U_{d,m}$ is the matrix consisting of the first d columns of the eigen matrice U_m. And U_m is obtained from the covariance matrix calculated from the L_t samples by SVD. GradiVeQ compresses the original gradient g into $U_{d,m}^{\top}(g-\mu_m)$.

PowerSGD [51] uses two low-rank matrices P and Q to approximate the original gradient matrix by PQ^T. The computation of P and Q is based on one power iteration and the Gramm–Schmidt orthogonalization. In addition, the authors extend it to decentralized scenarios [52]. The low-rank decomposition with power iterations is used to compress the model differences between neighboring nodes.

4.5. Summary and comparison

In this subsection, we summarize and compare aforementioned sparsification algorithms. Generally, a sparsification algorithm develops a dedicated rule to directly or indirectly filter the gradient elements and transmit them together with their corresponding indices. The difference from the compressed gradient and the original one should be as small as possible in terms of their directions and magnitudes. As the rule of thumb, the large elements in a gradient are retained while the smaller elements are discarded in most of the sparsification algorithms. The major differences among them are summarized in four aspects.

(1) *Element-wise* sparsification versus *holistic* sparsification.

The mainstream sparsification algorithms belong to the element-wise genre. They treat a gradient element as an atomic unit, and merely select a subset of gradient elements without modifying their values. The representative element-wise algorithms include Random-k, Top-k, DGC, etc. The holistic sparsification is featured with two approaches, sketch mapping and matrix decomposition. The sketch mapping makes use of a sketch data structure to transform the gradient elements into corresponding hash values, and hence the quantity of elements can be greatly reduced. The compression ratio of sketch mapping is subject to the size of the sketch data structure, but it can recover the global top-k gradient elements due to its linearity. Singular value decomposition is used to approximate the matrix-form gradient with a set of condensed matrices. Once being received by the server, they can recover all the gradient elements with gentle errors. The computational overhead of holistic algorithms are much larger than their element-wise counterparts.

(2) *Probabilistic* sparsification versus *deterministic* sparsification.

The probabilistic methods, as the name indicates, select gradient elements randomly, and representative ones include Random-k, GradSparse and Probcomp-LAPC. In particular, the latter two methods assign larger probabilities to the gradient elements with large magnitudes. The deterministic sparsification methods usually choose a threshold for pruning the gradient elements, or even sort them at a descending order. For example, DGC, threshold-v, Top-k and rTop-k all belong to the deterministic sparsification. The probabilistic methods demand the minimum computational complexity in contrast to the deterministic methods. These two types of methods are usually combined to balance the computational complexity and the precision of the sparsified gradient.

(3) Treatment on *residuals*.

The difference between the sparsified and the original gradients is called the *residual*. Although the residual is nonessential to the model update in one iteration, its accumulation over multiple iterations may influence the direction and magnitude of the gradient considerably. Simply discarding the residual impairs the convergence speed, and even cannot guarantee the exact convergence toward optimality under canonical strongly convex and smooth conditions in theory. To cope with this challenge, error accumulation compensation is presented for gradient sparsification, especially when the sparsity ratio is high. Most of the up-to-date algorithms tacitly approve this technique.

(4) *Complexity* order of compression.

We hereby only refer to the computational complexity that possibly prolongs the wall-clock time of an iteration, especially when the bandwidth is large and the compression ratio is high. Nominally, probabilistic sparsification algorithms such as Random-k have the lowest complexity order, i.e. $\mathcal{O}(k)$, that is independent of the gradient size. However, generating a random index is time-consuming such that Random-k is not a *de facto* efficient approach. The sorting algorithms, featured by Top-k, incur a much higher complexity order $\mathcal{O}(d\log k)$ on average using Quick Sort (despite of numerous sorting algorithms, Quick Sort is taken as the baseline) in a gradient with d elements. The threshold algorithms is of moderate complexity order that usually takes the form $\mathcal{O}(d)$. The actual computation time is pertinent to the threshold estimation that constitutes their core contribution and determines the running time well. The computation overheads of sketch mapping and matrix decomposition are much higher than the aforementioned approaches.

We summarize our comparisons in Table 1.

5. Communication quantization

Communication quantization reduces communication overhead with a lower accuracy to express each element. The bit width to represent each element is shortened. The communication quantization methods can be divided into two types. The first type is truncated expression that directly reduces the number of bits for each element from 32 bits to 16 bits, 8 bits or even 1 bit, as shown in Fig. 6(a). The second type is Code-Book mapping that turns continuous element values into certain quantization levels with corresponding discrete values.

The Code-Book mapping method typically involves deflating the original element values to a range between 0 and 1 before mapping. During communications, each worker transmits a sign vector, a scaling

Table 1
Comparison of different sparsification algorithms.

Algorithm	Element-wise or Holistic	Probabilistic or Deterministic	Error compensation	Complexity order of compression
Random-k	Element-wise	Probabilistic	✓	$\mathcal{O}(k)$
GradSparse	Element-wise	Probabilistic	×	Greedy algorithm
Probcomp-LAPC	Element-wise	Probabilistic	×	$\mathcal{O}(d)$
Top-k	Element-wise	Deterministic	✓	$\mathcal{O}(d \log k)$
rTop-k	Element-wise	Deterministic	✓	$\mathcal{O}(d \log k)$
gTop-k	Element-wise	Deterministic	✓	$\mathcal{O}(d \log k)$
ScaleCom	Element-wise	Deterministic	✓	$\mathcal{O}(d \log k)$
LGC	Element-wise	Deterministic	✓	$\mathcal{O}(d \log k)$
Threshold-v	Element-wise	Deterministic	×	$\mathcal{O}(d)$
GradDrop	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
DGC	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
RedSync	Element-wise	Deterministic	✓	$\mathcal{O}(logd + d)$
MSTop-k	Element-wise	Deterministic	×	$\mathcal{O}(d)$
EGC	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
SDAGC	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
Gaussian-k	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
SIDCo	Element-wise	Deterministic	✓	$\mathcal{O}(d)$
Sketched-SGD	Holistic	Deterministic	✓	$\mathcal{O}(d)$
FetchSGD	Holistic	Deterministic	✓	$\mathcal{O}(d)$
GradiVeQ	Holistic	Deterministic	×	$\mathcal{O}(d^3)$
PowerSGD	Holistic	Deterministic	×	Power iteration

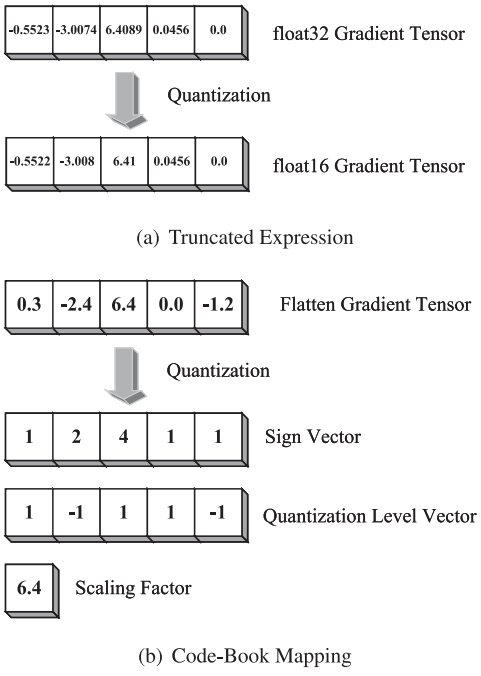


Fig. 6. Gradient quantization.

factor and a vector consisting of many quantization levels with a few bits, as shown in Fig. 6(b). It is worth noting that quantization methods cannot achieve a very high compression ratio because each element consumes at least one bit. If each original element takes 32 bits, quantization methods can achieve the highest compression ratio of $\frac{1}{32}$.

5.1. Truncated expressions

Various truncated expressions mainly differ in how many bits are used to represent a gradient element. FP16 [53] and 8-BIT APPROXIMATIONS [54] use 16 bits and 8 bits to represent each element, respectively. The extreme cases of truncated expression are 1-Bit SGD [55] and SignSGD [56], both of which adopt a single bit to represent each element.

FP16 [53] directly changes the precision of gradient elements from full-precision (float32) to half-precision (float16) by type-casting before transmission. The range of float32 is much different from that of float16, which might lead to numerical overflow. To avoid overflowing, FP16 reduces the range of gradient values by loss scaling in each iteration, and then the same scaling factor will be used to recover gradient elements.

8-BIT APPROXIMATIONS [54] uses 8 bits to represent each element, achieving a 4× compression. The representation of a floating point number includes sign part, exponent part and mantissa part. 8-BIT APPROXIMATIONS adopts a dynamic 8-bit representation without fixing the number of exponent bits and mantissa bits. It uses a flag bit to distinguish the exponent part from the mantissa part. Flag bit is the first '1' after the sign bit. The number of '0' between the sign bit and flag bit represents the exponent part, and the part after the flag bit represents the bifurcated dichotomous tree on the interval (0.1, 1). The method approximates large absolute values with small errors while retaining the ability to approximate small absolute values.

1-Bit SGD [55] uses a single bit to represent each element. It encodes elements whose values are less than the defined threshold as '0' and other elements as '1'. In the decompression process, '0' and '1' is reconstructed to the mean value of negative and non-negative local gradients. To further reduce the quantization error, 1-Bit SGD uses the error accumulation compensation technique.

SignSGD [56,57] also uses a single bit to represent an element, but only transmits the sign information. SignSGD quantizes the negative element to '0' and the non-negative element to '1'. In other words, it recovers quantized elements to either −1 or 1. SignSGD uses the majority voting to decide the sign of aggregated gradients. This method can tolerate packet losses during communications and adversarial attacks with fewer than half of attack workers. **SignNUM** [56] incorporates the momentum method into SignSGD. In addition, in order to reduce quantization error, **EFSignSGD** [58] combines with the error accumulation compensation technique with SignSGD. It transmits both the sign information and the value information together with the scaling factor.

LE-SignGD [59] further introduces a delayed aggregation technique to determine whether to skip the current communication or not. It defines a per-communication descent, i.e., the ratio of the one-step descent in SignGD to the total communication traffic in the cluster. LE-SignGD aims to obtain the largest per-communication descent by selecting a subset of workers in each communication round. If a local gradient is small enough, the corresponding worker will skip the communication of this gradient.

Adam has shown a superior convergence speed than SGD in a variety of deep learning tasks. However, it is hard to combine the compression and such non-linear gradient-based optimization algorithms together. Although error compensation is a common and effective technique in compression, it cannot be directly applied to Adam. There are two auxiliary variables, the variance term and the momentum term, in Adam. The compression error from the update of variance term contains a quadratic term, which cannot be eliminated with training. Thus, the compressed variance term cannot be estimated accurately. It is also required in the update of the momentum term to scale the compression error. To solve these problems, Tang et al. proposed **1-bit Adam** [60] based on the observation that the variance term of Adam becomes stable with training. 1-bit Adam first trains the model with the vanilla Adam as a warm-up. The variance term is fixed once it becomes stable. Then, the one-bit compression with error compensation (i.e., sign compression) is performed in the momentum term.

5.2. Code-Book mapping

Code-Book mapping can be divided into uniform and non-uniform methods based on the selection of quantization intervals. The former has the same interval value between two adjacent quantization levels; the latter has different interval values, and is more advantageous in expressing the distribution of gradient elements.

5.2.1. Uniform quantization

QSGD [61] maps continuous elements in g via random rounding to fixed discrete values with the scaling factor pertinent to the original gradient.

$$C_{QSGD}(g_i) = \|g\|_2 \cdot sign(g_i) \cdot \xi_i(g, s).$$

Here, $\|g\|_2$ is the L2-norm of the original tensor representing the scaling factor, and is used for both quantization and dequantization. $sign(g_i)$ is the sign function, indicating the sign value of the original element g_i; s is the total number of quantization levels, indicating that the range (0, 1) is equally sliced into s levels; $\xi_i(g, s)$ is a random variable defined by

$$\xi_i(g,s) = \begin{cases} \frac{l}{s} & 1-p \\ \frac{(l+1)}{s} & p = s\frac{|g_i|}{\|g\|_2} - l. \end{cases}$$

where l $(0 \le l < s)$ is an integer satisfying the condition of $\frac{|g_i|}{\|g\|_2} \in [\frac{l}{s}, \frac{(l+1)}{s}]$. QSGD ensures the unbiased quantization by introducing random variables. **ECQ-SGD** [62] applies the error accumulation compensation technique to QSGD to further reduce the negative impact of quantization error.

TernGrad [63] also performs quantization by a scaling factor related to the original tensor. Unlike QSGD, TernGrad specifies the quantization of all elements to only three levels $\{-1, 0, 1\}$, and its quantization results of tensor g can be expressed as

$$C_{TernGrad}(g_i) = \|g\|_\infty \cdot sign(g_i) \circ b.$$

The quantization method is similar to that of QSGD. The difference is that the scaling factor uses the maximum of the absolute value (infinite norm $\|g\|_\infty$) in the original elements where $\circ$ denotes the Hadamard product. Each element b_i in b obeys Bernoulli distribution independently, i.e.,

$$\begin{cases} P(b_i = 1|g) = \frac{|g_i|}{\|g\|_\infty}, \\ P(b_i = 0|g) = 1 - \frac{|g_i|}{\|g\|_\infty}. \end{cases}$$

Similarly, TernGrad introduces random variables to ensure quantization results unbiased. Because TernGrad has only three quantization levels, the vast majority of the elements will be quantized to 0 when the scaling factor is too large. Typically, most parameters will be changed slightly while only a small subset of them is updated significantly, resulting in a significant training variance. Therefore, TernGrad chooses the scaling factor layer-by-layer and performs the layer-wise trivalorization. Gradient clipping method is used to approximate the normal distribution of elements in order to prohibit the training variance.

In **DIANA** [64], both the workers and the server maintain a memory initialized to arbitrary values. Each worker performs a random trivalorization of the difference between the local gradient and local memory. The server collects the quantized differences from all the workers. The completion of the gradient aggregation relies on these collected differences and the memory in the server. The local memory in each worker is updated by its quantized difference. The server uses the aggregation result of all the differences to update the memory. DIANA can be combined with the Nesterov acceleration [65] and FISTA [66] in **ADIANA** [67]. **CANITA** [68] combines the acceleration algorithm ANITA [69] with DIANA.

In **MARINA** [70], each worker performs communication compression with some probability. Some workers quantize the difference of local gradients in two consecutive iterations and send it to the server. The remaining workers transmit the original local gradient directly. MARINA leads to a biased global gradient estimation. It can help the model to jump out of the local optimum trap in training.

3LC [71] is a three-valued quantization with lossless quadratic coding and zero-run coding. In the three-value quantization process, 3LC sets a sparse multiplier for scaling to control the sparsity of the quantization results. A simple rounding function $Round()$ is used for trivalorization in this process. The biased quantization from the rounding function needs to be compensated by the error accumulation compensation. Inspired by the quadratic polynomial expression, 3LC performs a lossless fixed-length encoding called quadratic encoding in a group of five trivialized results. In addition, 3LC also performs lossless zero-run encoding by exploiting the sparsity of the quantization results to recompress consecutive zero values. Therefore, 3LC can achieve a very high compression ratio.

FedPC [72] proposes a synchronous training algorithm that can preserve privacy while improving communication efficiency through ternary quantization in the worker–master architecture. The dataset, learning rate, batch size and optimizer used for local training in each worker are private information, which is invisible to the master and other workers. In addition, the size of workers' datasets is heterogeneous. FedPC deploys the algorithm on both the master and workers. At the beginning of each training epoch, all workers train the global model from the master locally to get a local model instance and an evaluated cost (e.g., loss function value). Each worker sends its cost to the master and waits for the master's command. The master selects the best model instances by means of a goodness function. The function jointly considers the size of a local dataset and the cost from workers. The selected worker sends the entire model instance to the master, and the remaining workers trivializes their model instances according to the change in models before communication. The global model update is performed by the master after receiving the information from all workers.

For the goodness function in the first training round, the master selects instances from workers carrying the lowest cost per data sample. Since the second training round, the master tends to select instances from workers with both a large dataset size and a large reductions of cost compared to the previous training round. After that, the master sends the command to all workers. For the trivalorization in the first training, each worker calculates the variation between the local model instance and the global model randomly initialized by the master. It is compared with the magnitude of local learning rate. The elements with large changes are quantified as +1 or −1. Whereas, the rest are quantified as 0. Starting from the second training round, each worker keeps the global model of the first two training rounds locally. The model instance is quantified based on the degree of the change in two model updates.

It is worth noting that FedPC consumes a certain amount of memory resource at both master and workers. Each worker needs to save the global model of the previous two training rounds. The master needs to store the cost of each worker in the previous training round and the corresponding size of a local dataset, which puts much pressure on storage when the number of workers is huge. Besides, due to the heterogeneity of workers' private information, they vary in the time spent in a single training round. There may exist workers which are on hold for a long time with a low resource utilization.

With 2-bit quantization, **CD-SGD** [73] designs a special update rule so that compressed communication and local updates can be carried out in fully parallel. Each worker updates the model with the gradients based on its local parameters. At the same time, the local gradients are quantized and sent to the servers. The communication and compression overhead can be largely hidden. When receiving the global parameter, the workers update their local model. K-step correction is used to transfer the full gradients every K iterations for correcting the direction of model update.

FedPAQ [18] and **PQASGD** [74] are composite compression that combine periodic communication and unbiased quantization to improve training efficiency. After each worker updates its local model for multiple rounds on the basis of the synchronized global model, it compresses the change between them for transmission. The parameter server performs the parameter synchronization after receiving all the compressed changes, and proceeds to the next periodic communication. FedPAQ allows the partial participation of workers. During each period, only a fraction of workers performs model training and communication.

LAQ [75] combines gradient quantization and lazily uploading. The quantization of the gradients depends on its relation to the previous quantized gradients. Each gradient element is mapped to the nearest discrete point in a uniform Code-Book centered on the previous quantized element with their difference as the radius. There is a rule for uplink communication in all the workers. Whether or not uplink communication is performed depends on the difference between the quantized gradients at current iteration and at the last uplink communication. The worker pushes the quantized information to the servers only when the difference is greater than a certain threshold. To prevent the problems caused by gradient staleness, LAQ sets a maximum skipping period. If a worker skips the uploading a certain number of times, it will be forced to participate in the next communication.

ATOMO [76] is a matrix-atomization-based compression method that represents the gradient matrix g as a linear combination of simple components (atoms) in the inner product space. The compression result is $g = \sum_{a\in\mathcal{A}} \lambda_a a$ for some sets of atoms $\mathcal{A}$ with $\|a\| = 1$. The objective is to minimize the variance in the atomic basis and keep it as an unbiased estimator of the original gradient.

AdaQS [77,78] uses the MSDR value of gradient elements as a metric to determine the quantization ratio of QSGD. MSDR can reflect the "signal-to-noise ratio" of gradient elements. When MSDR is less than a certain percentage of its historical value, the quantization level is doubled to suppress noises. Meanwhile, the historical value of MSDR is updated accordingly.

Oland et al. dynamically select the number of quantization levels for each layer by using the root mean square value of the layer gradients [79]. **MQGrad** [80] adopts a reinforcement learning based approach to determine the number of quantization levels by learning the change of the training loss. However, these two methods incur huge compression overhead, thus inefficient in practice.

AQG [81] determines the number of quantization levels according to the difference between element values obtained by quantization of the current iteration and that of the last iteration. A precision selection criterion is proposed to find the most appropriate quantization degree. The elements with smaller updates can be quantized with a fewer number of bits when the preset upper limit of bits is not exceeded. This specific process is to divide the difference into $2^b - 1$ intervals evenly based on the number of bits b, and find the fixed value of that interval closest to the original element. AQG designs Augmented AQG to accommodate client drops. Augmented AQG draws on a gradient scaling method to ensure the unbiasedness of gradients.

UVeQFed [82] proposes to use subtractive dithered lattice quantization by assuming that the scalar quantization is less effective than the vector quantization. This method requires all the workers to use the same compression operator with the same random seed. It does not require any prior knowledge about the element distribution. UVeQFed first initializes a non-singular square matrix as a lattice generator matrix, which can be considered as a vector version of the Code-Book. The lattice point is represented by a linear combination of the columns of the lattice generator matrix. Lattice quantizer maps the original vector to the closest lattice point to the original value. Since all vectors nearest to a lattice point are quantized to this point, it is called the lattice quantization.

The specific quantization process of UVeQFed is as follows. Each worker scales the model update according to its norm locally. The scaled model update is split into M copies according to the dimension of the lattice generator matrix. Then each worker performs random dithering of these model updates with a shared random seed. Then, the lattice quantization is used to compress and generate a digital codeword vector, and this vector is sent to the server which decompresses the model update by the inverse operation of the above compression process and performs the aggregation.

5.2.2. Non-uniform quantization

After scaling with the infinite norm or L2-norm of the original tensor, the normalized gradient distribution is not uniform [83]. Uniform quantization cannot well capture the nature of uneven gradient distribution. Most gradient elements will be mapped to the same quantization level, leading to a large quantization variance. This motivates the design of non-uniform quantization, given the knowledge of the gradient distribution. Non-uniform quantization usually falls in the scope of the Code-Book mapping.

NC (Natural Compression) [84] uses uneven quantization levels to round the value of an element to one of its two nearest integer powers of two. It is also an unbiased communication quantization method. The specific quantization process is shown as below

$$C_{NC}(g_i) = \begin{cases} sign(g_i)\cdot 2^{\lfloor \log_2 |g_i| \rfloor} & p = \dfrac{2^{\lceil \log_2 |g_i| \rceil} - |g_i|}{2^{\lfloor \log_2 |g_i| \rfloor}} \\ sign(g_i)\cdot 2^{\lceil \log_2 |g_i| \rceil} & 1-p. \end{cases}$$

The advantage of NC is that there is no additional computational overhead other than the randomization process. It is essentially equivalent to removing the mantissa part from the binary expression of gradients and is "naturally" compatible with binary floating-point types.

NUQSGD [83] changes the quantization intervals of QSGD from a uniform partition between 0 and 1 to a dichotomous partition. If the total number of quantization levels is $s+1$, the quantization level of NUQGSD is $\left\{0, \frac{1}{2^s}, \frac{2}{2^s}, \ldots, \frac{2^s-1}{s}, 1\right\}$. The main idea of NUQSGD is to reduce the quantization error and control the variance by dividing small gradients more finely. The new quantization levels in NUQSGD is concentrated near zero when the total number of quantization levels increases. It is not very effective to the quantization of large gradient elements [85].

APoT [85] is a non-uniform parameter quantization algorithm designed for the bell-shaped and long-tailed parameter distribution. It aims to solve two challenges. One is how to design quantization levels efficiently. When designing the non-uniform quantization, there are still two problems. On the one hand, non-uniform quantization like NC and NUQSGD overlooked the mapping of dense small weights and sparse large weights. Higher resolution should be given to the weights with larger number or more contributions to update in order to make the model convergence faster. On the other hand, general non-uniform quantization incurs more compression overhead than uniform quantization because they are usually not hardware friendly. Another challenge

Table 2
Comparison of different quantization algorithm.

Algorithm	Category	Uniform or Non-uniform	Dynamic quantization
FP16	Truncated expression	–	×
8-Bit	Truncated expression	–	×
SignSGD	Truncated expression	–	×
QSGD	Code-book mapping	Uniform	×
TernGrad	Code-book mapping	Uniform	×
DIANA	Code-book mapping	Uniform	×
3LC	Code-book mapping	Uniform	×
FedPC	Code-book mapping	Uniform	×
CD-SGD	Code-book mapping	Uniform	×
FedPAQ	Code-book mapping	Uniform	×
LAQ	Code-book mapping	Uniform	×
AdaQS	Code-book mapping	Uniform	✓
MQGrad	Code-book mapping	Uniform	✓
AQG	Code-book mapping	Uniform	✓
UVeQFed	Code-book mapping	Uniform	×
NC	Code-book mapping	Non-uniform	×
NUQSGD	Code-book mapping	Non-uniform	×
APoT	Code-book mapping	Non-uniform	×
ALQ	Code-book mapping	Non-uniform	✓
MUSCS	Code-book mapping	Non-uniform	×

is how to choose a suitable threshold for clipping. Before quantization, clipping is applied to restrict the original weights to a certain range. However, the range of clipping and the resolution of quantization are antagonistic. A larger range preserves more weights, but may reduce the resolution of subsequent quantization. An inappropriate clipping may cause the network to converge at a slower rate, or even fail to converge. How to automatically find the appropriate threshold for clipping in training is still an unresolved problem. APoT proposes constructing quantization levels based on the sum of basic quadratic power quantization levels. Only the bit-wise shift operation is in the quantization process. It is computationally cheap, which only takes 1 clock cycle in modern CPU. This greatly reduces the compression overhead. The weight normalization is used on the parameters before quantization to make them more stable and consistent for mapping. In addition, APoT designs a Reparameterized Clipping Function to compute a more accurate gradient.

Both **ALQ** and **AMQ** [86] are adaptive quantization methods for adjusting the intervals of different quantization levels. The interval of quantization levels is determined by minimizing the expected variance or the normalized expected variance of quantization. ALQ minimizes the excess variance of quantization for a given gradient distribution estimate. Contrary to ALQ, AMQ uses exponential quantization levels.

MUCSC [87] uses a clustering method to determine the discrete values after quantization. It divides original elements into several clusters. The centroid value of each cluster is considered as the quantization value of the elements in this cluster. The results of MUCSC are unbiased estimation of original elements. An enhanced algorithm, namely **B-MUCSC**, is proposed to improve the compression degree of MUCSC. The major difference of B-MUCSC is that it manually defines a super cluster including elements closest to zero.

5.3. Summary and comparison

In this subsection, we summarize and compare the versatile quantization algorithms. Gradient quantization aims to reduce the communication load by lowering the precision of each element. With random rounding, most quantization methods are unbiased, and the convergence of quantization approaches are guaranteed without the need of error accumulation compensation. Note that each gradient element is double-precision (i.e. 32 bits), the compression ratio is no more than 32 times, and at the maximum ratio each elements uses a single bit to represent its sign. The major differences of existing quantization approaches are summarized in three aspects in Table 2.

(1) *Truncated expression* versus *Code-book mapping*.

One way to quantize a gradient element is its truncated expression that directly converts the double-precision type to the bit-width of the certain binary representation. The other is Code-book mapping that turns a continuous gradient element into one of the predetermined discrete interval values. Only several bits are needed to represent an interval and hence the gradient element. The code-book mapping in general outperforms the truncated expression because the latter is harmful to the compression of large gradient elements.

(2) *Uniform* mapping versus *Non-uniform* mapping.

The spirit of code-book mapping is to represent a gradient element with less bits. Uniform mapping evenly partitions the range between the maximum and the minimum values of gradient elements. QSGD and TernGrad belong to this category. Measurement studies show that the distribution of gradient elements resembles a bell shape with a long tail, other than a flat uniform distribution. Hence, the uniform mapping may introduce large quantization errors, impairing the convergence of model training. Non-uniform mapping usually configures the intervals according to the gradient element distribution in order to minimize the expected quantization errors. NC, NUQSGD, APoT, ALQ and MUCSC are non-uniform mapping methods that only differ in the specific techniques used for computing the intervals.

(3) *Dynamicity* of quantization.

The gradient distribution is dynamically changing as the training moves forward. An ideal quantization algorithm should be non-uniform, and keep track of the changing distribution of gradient elements. Methods like QSGD, TernGrad, NC and NUQSGD belong to static quantization. Their quantization levels or intervals are fixed after the training starts. The dynamic quantization methods adopt different approaches to adjust the intervals. AdaQS and MQGrad adaptively increase or decrease the total number of quantization levels, while AQG adjusts the precision of the current quantization on the basis of the previous quantization. The interval of ALQ is determined by solving a minimization problem of reducing the variance of quantization.

6. Hybrid compression

The essence of sparsification is to reduce the number of elements and the essence of quantization is to express each element with a fewer number of bits. These two compression methods are orthogonal in their core design, implying that they can be combined as hybrid methods. The effectiveness and feasibility of the hybrid compression has been proved theoretically in [88].

Sparse one-bit quantization [89] is a hybrid compression method. Given a threshold υ, the elements with values between $-\upsilon$ and υ are discarded, while the remaining elements greater than υ and less than

$-\upsilon$ are quantified as '1' and '0', respectively. They are decoded as υ and $-\upsilon$ when decompressed. A user-defined threshold is hard to choose in practice, and thus adaptive threshold-υ dynamically sets the positive and the negative thresholds [90] with a predefined ratio to select the positive and negative gradient elements among the total elements. The positive threshold υ^+ and negative threshold υ^- that satisfy the sparse ratio are estimated by random sampling.

Qsparse-local-SGD [88] is a hybrid compression method that first performs a sparsification operation on original gradient elements to exclude unimportant ones. Then, a quantization operation is conducted to further improve the compression ratio. Since both sparsification and quantization will amplify the variance of compressed gradients, the hybrid method can enlarge the variance proportionately. To prohibit variance explosion, Qsparse-local-SGD scales down the results of hybrid compression so that the variance bound is limited. Besides, Qsparse-local-SGD applies the error accumulation compensation technique to diminish its compression error.

SBC [91] combines communication sparsification, binary quantization, and Golomb coding techniques. It also adopts compensation techniques such as error accumulation compensation, momentum correction, warm-up training and momentum factor masking to improve model accuracy. SBC first sets a key fraction parameter p. All elements except the top-p fraction and the bottom-p fraction are reset to zero. Then, SBC calculates the mean values μ_+ and μ_- for all remaining positive and negative elements, respectively. If $|\mu_+| > |\mu_-|$, SBC sets all negative elements to 0 and all positive elements to μ_+, and vice versa. Therefore, SBC binarizes all the gradient elements into 0 and μ_+ or 0 and μ_-. SBC only transmits non-zero elements and their indices. To further compress element indices, SBC employs Golomb encoding to only transmit index distances between these non-zero elements.

The nutshell of **HGC** [92] is to avoid gradient staleness with sparsification and further improve the communication efficiency by quantization. The sparsification with low sparsity can maintain more information for less accuracy loss and the quantization is used to achieve a higher overall compression ratio. HGC first estimates a reasonable threshold from the mean values of original gradients that will be used to perform the sparsification. Then, it employs a modified QSGD gradient quantization method to reduce the number of bits to express each gradient element. To improve accuracy, an error buffer is enabled locally for error accumulation compensation.

After sparsifying gradients with the Top-k, **FEDZIP** [93] further quantifies sparsified gradients with the k-means clustering method. The advantage of k-means clustering for quantization is that the distribution of gradient elements can be captured by k-means. Moreover, FEDZIP performs Hoffman coding on k-means clustering results, i.e., the centroid values of k clusters. However, k is a hyperparameter that should be determined based on prior information. Thus, it is difficult to automatically adapt k with different models.

Variance-based Compression [94] is also a hybrid method of sparsification and quantization. In the gradient sparsification step, it filters gradient elements based on both gradient magnitudes and their variance. When the absolute value of a gradient is smaller than its variance at the data point, the gradient element is regarded as "fuzzy" and dose not participate in this round of communication. After obtaining sparsified gradients, Variance-based Compression quantifies each gradient with 4 bits (1 sign bit and 3 exponent bits), and quantifies each index value with 28 bits. Thus, it consumes 32 bits in total to represent the value and the index of each selected gradient element.

AdaComp [95] is a hybrid compression method based on the error accumulation compensation technique. It first divides gradient elements (compensated by residuals) into several bins uniformly, and identifies the maximum value (in terms of the absolute value of gradients) in each bin. A gradient is important and will be included in the communication if this gradient multiplied by a scaling factor is greater than the maximum value of its bin. Selected important gradients will be further quantizated before transmission.

MGC [96] is a hybrid compression method that combines a gradient sparsification operation with a three-value quantization operation. In order to preserve the training accuracy, MGC divides the entire training process into three different phases: warm-up phase, stable phase and best-effort phase. At the warm-up phase, MGC adopts a low degree of sparsification and proposes automatic sparsity adjustment to change the sparsity in accordance with unstable parameters. At the stable phase, the neural network gradually stabilizes with much smaller changes of parameters. The gradients become monotonic and stable compared with those at the warm-up phase, implying that high sparsity is applicable. At the best-effort phase, the volatility of the loss function and the parameter size tends to weeny [63]. The quantization step will produce the errors in most gradients, leading to an inaccurate direction of model update. Therefore, the quantization recession is proposed to terminate quantization in the best-effort stage.

DeepReduce [97,98] compresses both the indices and values of gradients filtered by sparsification. It uses a probabilistic data structure, Bloom Filter, to compress the indices. The mapping conflicts in query phase of Bloom Filter may lead to an intolerable error on the reconstruction of sparse gradients. A no-error method P0 is proposed to reduce the errors. All the mapped indices and their corresponding values will be transmitted. Since P0 may cause the communication volume to increase, DeepReduce designs P1. P1 randomly selects a specified number of entries from the mapped indices and transmits them and their values. However, there are still some errors in P1. In view of this, DeepReduce further proposes P2 to balance the communication volume and the mapping errors by performing the random selection strategy among the indices with conflict. Besides, a nonlinear approximation method is designed for value compression. It uses a spline with free knots to split sorted gradients into segments. Each segment is fitted to determine the values of these free knots so that the gradients can be quantized.

SketchML [99,100] first creates a quantile sketch to store all the non-zero gradients. These gradients are divided equally into different buckets. Gradients in each bucket are encoded into the corresponding bucket index, and the value is quantified as the mean value in the bucket. SketchML further compresses the binarized bucket index based on MaxMinSketch. In the insert phase of MaxMinSketch, it selects the minimum value after mapping to update Sketch. In the query phase, the maximum value among candidates is selected as the estimation result. In addition, to reduce the sign reversing problem that MaxMinSketch may bring, SketchML compresses positive and negative gradients separately. To overcome the gradient vanishing problem, SketchML further groups the gradients and compresses them separately.

6.1. Summary and comparison

In this subsection, we summarize and compare the aforementioned hybrid compression algorithms. The sparsification method and the quantization method are orthogonal so that the hybrid compression can potentially take the best of both worlds. The hybrid compression is able to achieve higher communication reduction by transmitting less gradient elements and transmitting less bits for each element. A summary of these algorithms is shown in Table 3.

One can see from the table that in hybrid compression, gradients are sparsified first and then quantized. It is helpful to preserve important elements. Furthermore, a few algorithms such as SBC, Variance-based compression and Deepreduce compress the indices with lossless encoding to further improve the compression degree. MGC dynamically adjusts the combination of sparsification and quantization in different training stages.

Table 3
Ideas of combination for different hybrid compression algorithms.

Algorithm	The idea to combining sparsification and quantization	Compression methods included
Sparse 1-bit	Sparsification followed by quantization.	Preset threshold-v and 1-bit SGD on values
Qsparse	Sparsification followed by quantization.	Top-k/Random-k and stochastic quantization on values
SBC	Sparsification followed by quantization.	Top-k and adaptive 1-bit SGD on values Golomb encoding on indices
HGC	Sparsification followed by quantization.	Adaptive threshold-v and QSGD on values
FEDZIP	Sparsification followed by quantization.	Top-k and k-means clustering on values
Varianced	Sparsification followed by quantization.	Sparsification and 4-bit truncated expression on values 28-bit quantization on indices
Adacomp	Sparsification followed by quantization.	Adaptive sparsification and EFSignSGD on values
MGC	Dynamic combination with training.	Different combinations on different stages of training
DeepReduce	Approximation on sparse gradients	A nonlinear approximation on sparse values Bloom Filter on indices
SketchML	Sketch-based quantization on no-zero gradients.	MaxMinSketch quantization on no-zero gradients

7. Context specific compression

Compression algorithms can be summarized from the context specific compression perspective. It mainly refers to the customized design of compression algorithms to accommodate different communication architectures and adapt with different environments. In this section, we describe the technical details of existing context specific compression methods. Context refers to the communication architecture, resource environment and application, etc. Context specific compression is designed to improve the training efficiency by adapting to these contexts with compression. We describe these compression methods as frameworks to meet specific context conditions.

7.1. Decentralized compression framework

The distributed training stands for the communication pattern that each local worker uploads his local model and downloads the global model at every round. Distributed training can usually be divided into centralized training and decentralized training. In centralized training, the aggregation of local models into the global model is executed either at the centralized parameter server or through the All-reduce passing of fragmented parameter chunks over multiple slots. Centralized training is widely used because of its simplicity of deployment. The vast majority of compression algorithms introduced previously are designed based on centralized learning.

However, the parameter server tends to be a network bottleneck, especially when the network bandwidth is low and the number of workers is large. In addition, centralized training suffers from poor fault tolerance. Therefore, centralized training is often deployed in fast and stable network environments. In the network with limited bandwidth and unstable connections, decentralized training shows better performance. In decentralized training, the communication pattern is expressed as a graph in which each worker only exchanges its local parameters with its neighbors. Decentralized training removes the need for a central coordinator. It provides good robustness and reduces the bandwidth requirements between different machines. What is more, data security concerns as well as privacy issues are mitigated. Decentralized training also requires communication compression, but direct transmission of compressed parameters incurs compression errors, probably causing learning divergence [101]. In order to guarantee convergence, the decentralized compression framework must be dedicatedly designed.

Tang et al. proposed two different unbiased compression frameworks, **DCD-PSGD** and **ECD-PSGD**, for decentralization of low-bandwidth and high-latency networks [101]. The former compresses the difference of the locally updated parameters in two consecutive rounds. The latter compresses the extrapolation between the last two local parameters. When the data variance between workers is very large, DCD-PSGD converges lightly faster than ECD-PSGD. With guaranteed convergence, ECD-PSGD can achieve a higher degree of compression than DCD-PSGD.

Choco-SGD [102,103] is a decentralized framework that can combine unbiased and biased compression algorithms. It introduces an auxiliary variable that follows the variation of model parameters. The difference between model parameters and corresponding auxiliary variables are compressed. The auxiliary variables are updated by accumulating the compressed differences. This method gradually reduces the compression error as training proceeds.

DC-DGD [104] is a decentralized framework for SNR-constrained unbiased compressors. It can accommodate a wide range of general compression algorithms. DC-DGD compresses local estimated gradient differences in two subsequent iterations. No additional mechanism or hyperparameter is needed to guarantee the convergence of learning. Based on DC-DGD, **ADC-DGD** [105] amplifies the difference to effectively reduce the accumulation of compression errors.

To guarantee convergence, many decentralized compression frameworks require additional memory to store historical information or auxiliary variables. **Moniqua** [106], which require zero additional memory, is proposed for the case where the local memory of the worker is limited. It compresses and decompresses model parameters according to the similarity between workers in the consensus process and the *mod* operation. The value of x can be recovered only by the value of x mod 2θ and y when $|x - y| < \theta$:

$$x = (x \ mod \ 2\theta - y \ mod \ 2\theta) \ mod \ 2\theta + y.$$

For the compression algorithm C, Moniqua requires that the infinite norm of compression error is limited by a constant δ. The compression result of x is $C(x/(2\theta/(1-2\delta)) \ mod \ 1)$.

Decentralized learning is always deployed in a network with limited and heterogeneous bandwidth. Most of the current decentralized compression frameworks have overlooked the diversity and heterogeneity of bandwidth capacity among clients.

GossipFL [107] is designed for dynamic and heterogeneous bandwidth in decentralized training aiming to improve bandwidth utilization. It enables each client to only exchange a highly compressed model with a single neighbor. The neighbor of each client is dynamically selected in each communication round based on the bandwidth. GossipFL has two roles, coordinator and client, and the communication between them is very lightweight. The coordinator is responsible

for generating the Gossip Matrix and the communication timestamp matrix. It sends the Gossip Matrix and random seeds to all clients. At the end of each iteration, the coordinator collects the bandwidth and training information from each client. A client performs random sparsification on its local models with the random seed obtained from the coordinator, because the same-index sparsification is helpful to the consensus. Each client sends or receives the compressed model to or from the corresponding neighbor to perform a weighted average according to the Gossip Matrix. Afterwards, each client combines the averaged model with its own unexchanged part which can be regarded as a kind of error compensation operation performs. The SGD update is conducted with the local training data. Finally, the client sends the bandwidth and training information to the coordinator.

The coordinator generates the Gossip Matrix based on bandwidth and timestamp information. The connections of all workers can be analyzed by a graph. It tends to select edges (connecting workers) with high bandwidth on a sub-graph G^*, in which an edge connection exists if the edge bandwidth is greater than a threshold. However, always selecting the best match neighbor all the time cannot guarantee the connection of the graph and may lead to multiple isolated communication sub-graphs, breaking the model consensus of workers. To overcome this problem, the coordinator further builds a recently connected edge set RC, representing the set of edges selected in a recurrent short time window. In each iteration, the coordinator first determines whether RC forms a connected graph. If so, it selects the best match in G^*. Otherwise, the coordinator randomly select edges out of RC.

SwarmSGD [108] is an asynchronous communication compression framework. Each worker has a buffer that can be accessed by its neighbor nodes. At each time step, the worker that completes expected local updates randomly chooses a neighbor as the target worker. It reads the compressed model in the target worker's buffer and takes the average with its own compressed model. The buffers of the two workers are update with the average result. It should be noted that when reading the compressed model from buffer, the compressed model may have been rewritten by other workers.

Quantized Push-sum for Gossip [109] combines the communication compression with the Push-Sum algorithm in Gossip. The parameter compression is employed on the directed graph communication topology and the compression algorithms should be unbiased.

Sparse-Push [110] is a decentralized communication compression framework for time-varying directed graph. In order to guarantee the convergence, it combines the error accumulation compensation technique with the Push-sum algorithm for time-varying directed network [111]. Each worker compresses the compensated parameters and sends them and the auxiliary parameter bias weight in the Push-sum algorithm to its out-neighbors.

With the same communication volume, exchanging low-precision parameters multiple times in one iteration brings a faster convergence rate than exchanging high-precision parameters only once. According to this observation, **DeLi-CoCo** [112] combines the parameter compression with the Multiple Gossip optimization algorithm. After workers perform local updates, a fixed number of compressed Multiple Gossip is performed. The compressed Multiple Gossip includes compressed information exchange, error accumulation compensation and local consensus update.

C-GT [113] sets a state variable that tracks the local model parameters and the local auxiliary variables. Each worker compresses the difference between its model parameters (or auxiliary variables) and their corresponding state variables. The state variables are updated based on the sliding average method. It is worth mentioning that some general biased compression algorithms, such as the norm-sign compression, can be combined in C-GT. Theoretically, a linear convergence rate can be achieved on strongly convex and smooth objective functions.

Kajiyama et al. design a compression framework for Gradient Tracking [114]. The model parameters as well as auxiliary variables are encoded and decoded dynamically through a specific quantization. A linear convergence rate can be achieved in theory. However, the reliance on specific quantization makes the design very complex and the computational overhead large. The method is only effectively used on the weight-balanced directed network.

LEAD [115,116] sets a state variable to track the local model parameters. Before each round of communication, the difference between the model parameters and the state variable are compressed with unbiased algorithms. The state variable is updated based on the parameter values and the sliding average method. In particular, LEAD sets a gradient correction variable in local update to accelerate the convergence.

Decentralized compression frameworks that combine error accumulation compensation methods include **DeepSqueeze** [117] and the method in [118]. Each worker sets a local error cache to store the compression error generated in training. If the error accumulation compensation is applied to the entire parameter update, some dimensions of the consensus update will be incorrectly scaled up. The incorrect amplification will result in the final divergence. To address this issue, **ECSD-SGD** [21] separates the parameter update into a consensus update part and a gradient update part. It only compensates the error in the gradient update part where the compression error is generated.

7.2. Environment dependent compression methods

When deploying compression algorithms in practical systems, new challenges arise, such as the computational capacity constraint and the limited network bandwidth. Complex and large models used in distributed deep learning place high demands on both the computational power and the communication capabilities of the workers, while compression overhead also requires some computing resources. Poor load balancing can lead to unbearable resource consumption of the machine, which is detrimental to industrial applications and even more fatal for some mobile devices with limited battery life. When the communication medium used is a wireless channel, the transmission may be interfered, leading to further delays and errors [119]. Meanwhile, it is difficult to maintain a constant computation capacity and a fixed bandwidth at a worker during the entire training cycle. Therefore, the compression methods need to adapt to the runtime environments in order to genuinely reduce the wall-clock time of model training. We hereby survey the recent progresses on the environment dependent compression methods.

In a heterogeneous environment, the computational and communication capabilities of each worker are different. Capable workers may complete computation and communication early and wait for a long time for synchronization. To solve such problems, asynchronous compression framework **AsyLPG** [120] is proposed. It inherits the model parameters (snap variables) from the previous epoch to calculate the full-batch gradients. In each iteration, the servers use the compressed gradients from the worker and the full-batch gradients to update global parameters. One worker is randomly selected to compute the gradient based on its local gradient, snap variables and the compressed global parameters from servers. After that, the worker sends the compressed gradients to the servers. At the last iteration, the global model parameters are inherited as new snap variables to the next epoch.

Canonical compression frameworks usually assume a time-invariant computing or networking environment. However, the computing capability of a worker is usually uncertain and the network bandwidth is time varying either in a datacenter or in Internet. Such dynamics will cause the communication time hard to be predicted in the synchronous training, and cause the serious staled gradient problem in the asynchronous training. Thus, the well calibrated but fixed compression techniques will perform far below their desired objectives.

DC2 [121] contains a suite of five algorithms to control the sparsification ratio in response to bandwidth variation. The first algorithm, DA1, adjusts the sparsification ratio in proportion to the normalized difference between the instantaneous latency and the mean latency. As

a double-edged sword, the simplicity of DA1 leads to serious oscillations of the sparsification ratio especially when the bandwidth is highly dynamic. The second algorithm, DA2, defines a target latency of each iteration and keeps the instantaneous latency around it. An additive increase multiplicative decrease (AIMD) rule enforces this goal, which mimics the TCP congestion control mechanism. If the instantaneous latency exceeds the target, the sparsification ratio is halved, and it increases linearly otherwise. The third algorithm, namely DA3, substitutes the multiplicative reduction of AIMD by the linear additive reduction (AD). The fourth algorithm, DA4, adjusts the compression ratio according to the normalized gradient of delay changes. The last algorithm, DA5, introduces a threshold range on the basis of DA4, and adopts AIMD to adjust the compression ratio if the latency of an iteration is beyond this range. Then, DA5 is able to adapt to the bandwidth variation and avoid the drastic fluctuation of the compression ratio.

As training proceeds, the gradients will change. A fixed compression ratio may lead to under-compression or over-compression at different training stages. **ACCORDION** [122] designs a rule to judge the critical regimes in training according to the norm of the gradient. The precise communication among workers is needed only in the critical regimes [123,124]. According to these works, ACCORDION uses different compression ratios inside and outside the critical regimes. It performs more precise communication in the critical regimes.

Many compression frameworks are designed to address the difference in the computation and communication capabilities of single worker. They work to increase the overlap of computation time and communication time, thus reducing the overall training time.

LAGS-SGD [125] aims to maximize the overlap of computation and communication of each layer in the pipelined training. It sets the compression ratio according to the ratio of the computation time to the communication time of each layer in the model. In order to avoid the large amount of startup overhead caused by layer-wise compression in LAGS-SGD, **OMGS-SGD** [126] introduces the tensor fusion [127] before communication compression. It merges the gradients of successive layers together before the compression. The selection of merged layers is obtained by an optimization problem that minimizes the single iteration time. To hide the compression overhead and waiting time for tensor merging, the compression ratio is set according to the computation-communication pipeline.

In large-scale DNN training, hybrid parallelism has been widely used. The workers are divided into several groups in hybrid parallelism. The workers in each group store different parts of the DNN and all the groups are trained in data parallelism. The hybrid parallelism relieves workers from computational and storage pressure, but it introduces a larger communication overhead. In addition to each group of workers needing to send gradients to parameter server for aggregation, the workers in one group also need to exchange activation values and gradient information in both forward propagation and backward propagation.

According to the observation that the threshold corresponding to a certain compression ratio does not vary significantly across iterations, **DCT** [128] proposes a dynamic sparsification method with hard threshold comparison. The threshold setting can remain constant over a period of time (thousand iterations), which greatly reduces the compression overhead. DCT performs communication compression in data parallelism and model parallelism, respectively. For data parallelism, the gradients are sparsified layer-by-layer. For model parallelism, DCT performs sparsification on the activation values generated by each data sample separately before activation communication during forward propagation. The corresponding selected indices are directly used in the sparsified gradient communication in the backward propagation. DCT can be deployed in different applications, such as recommendation systems and natural language processing.

Most of aforementioned distributed compression algorithms are designed based on clustered workers. Next, we proceed to introduce applications of communication compression in wireless federated learning scenarios.

In the IoT edge-computing scenario, the communication and computational capabilities of low-power devices are always highly heterogeneous. It is difficult to execute a distributed training scheme designed for data center equipped with high performance computing and networking hardware to work well in such applications. Even though the FedAvg algorithm allows for distributed training at the network edge in the IoT framework, the huge upload overhead for communication and the high number of iterations to reach convergence are concerns.

CE-FedAvg [129] combines the distributed Adam algorithm with upload compression to reduce both the number of communication rounds and the total transmission volume to complete model training. Adam optimization is used at clients to accelerate convergence. The compression strategy at clients is to sparsify parameters and auxiliary variables in Adam and then quantize them. In the sparsification phase, parameter elements are selected according to their absolute values and their corresponding auxiliary variable elements are chosen. This design is based on the finding that if parameters and auxiliary variables are sparsified independently, the gradient explosion will occur soon due to obsolescence. For selected values, parameters and auxiliary variables are quantized separately. CE-FedAvg groups parameters by their sign values and quantizes them uniformly based on the maximum and minimum values in each group. The auxiliary variables are compressed by dubbed exponential quantization because of their large range of values and sensitivity to error. For selected indices obtained from sparsification, the lossless Golomb coding is used to further reduce the communication burden.

With the development of mobile edge computing, the computing power of modern smart mobile devices is expanded, which facilitates the application of federated learning in mobile networks. However, the practical deployment faces challenges. On the one hand, model training is resource-intensive and limited battery life hinders the use of mobile edge devices for complex model training and continuous learning. On the other hand, the mismatch between the communication volume, wireless bandwidth and the heterogeneity of communication environment lowers the gradient exchange efficiency.

To balance the energy consumption of local computing and wireless communication from the long-term learning perspective, **FL-LSGD-DB** [130] uses the energy consumption of edge devices as the metric to consider the energy efficiency of the whole training. It determines the total number of required communication rounds by the derived convergence bound and the modeled energy consumption for both communication and computation. FL-LSGD-DB builds an optimization problem with the goal of overall energy consumption minimization. A compression algorithm integrating the Benders decomposition and inner convex approximation is proposed to determine the compression ratio for every participant and the number of local computations.

From high-performance clusters to energy-limited IoT devices, communication efficiency becomes a more challenging bottleneck for distributed training. Compression is a common approach to reducing communication overhead, but aforementioned compression methods require careful tuning of the compression ratio before training is conducted. **CAT** [131] proposes an online strategy to adaptively adjust the compression ratio for each specific problem and gradient information. CAT models the communication cost and the improvement of objective function after compression based on the theoretical result of single-step descent. An optimization problem is established to maximize the ratio between the objective function improvement and the communication cost. The optimal compression ratio can be obtained by solving this optimization problem for each iteration.

In federation learning under wireless networks, the stochastic channel fading, limited bandwidth and the constrained computational power of devices can trigger a large learning latency. How to shorten this learning latency is an essential issue in wireless federation learning. Liu et al. shows that the learning latency can be reduced by balancing communication and computation cost [132]. The batch size and compression ratio are important parameters determining the computation

Table 4
Comparison of different decentralized compression framework.

Framework	Suitable topology	Compressor
DCD-PSGD	Undirected	Unbiased
ECD-PSGD	Undirected	Unbiased
Choco-SGD	Undirected	Unbiased and biased
DC-DGD	Undirected	Constrained by SNR
GossipFL	Undirected	Random sparsification
Moniqua	Undirected	Mod operation
swarmSGD	Undirected	Unbiased
QPush-sum	Directed	Unbiased
Sparse-Push	Directed and time-varying	Unbiased
DeLi-CoCo	Undirected	Unbiased and biased
C-GT	Undirected	General
LEAD	Undirected	Unbiased
ECSD-SGD	Undirected	Unbiased

and communication costs, respectively. Their work studies the impact of batch size and compression ratio on convergence rate, and how to balance the trade-off between the learning latency and the convergence rate. More specifically, an optimization problem is formulated to maximize the convergence rate for a given training latency constraint by jointly optimizing the batch size, compression ratio and spectrum allocation in the orthogonal frequency division multiple access and time division duplex wireless networks. Here, the convergence rate is formulated by the average L2-norm of gradients, and its connection with the compression ratio and batch size is derived theoretically. The optimization problem is further decomposed into two sub-problems, which can be solved by an iterative adaptive algorithm.

7.3. Summary and comparison

In this subsection, we summarize and compare these context specific methods. Context specific methods roughly fall into two categories: decentralized compression frameworks and environment-dependent algorithms.

In decentralized training, the model parameter is transmitted from each worker to its neighbor(s). Since unlike the gradient the model parameter is far from zero, direct transmission of compressed parameters is inappropriate. Therefore, the decentralized training calls for new compression framework to ensure convergence. Error accumulation compensation is the default method to guarantee convergence. The major differences among decentralized compression frameworks are summarized in two aspects in Table 4.

(1) Directed versus undirected communication *topology*.

The undirected topology allows the workers at the both ends of a link to exchange information bilaterally. In contrast, the information can only flow from one work to the other. Most compression frameworks consider the undirected communication topology. GossipFL is implemented on the undirected communication topology, but it can automatically adapt the topology after each iteration. QPush-sum and Sparse-Push can be used in directed communication topology. In particular, Sparse-Push can be deployed in the time-varying directed topology.

(2) Combinable communication *compressor*.

The compression algorithms are in general universally applicable to different learning algorithms. We hereby name a couple of special compressors, i.e. GossipFL and Moniqua, in which GossipFL can only use random sparsification, and Moniqua only uses the *mod* operator for compression. On the other hand, most learning frameworks (related to decentralized topology) only utilize unbiased compression operators, while several of them such as Choco-SGD, DC-DGD, DeLi-CoCo and C-GT hold for general unbiased and biased compressors.

Environment-dependent compression refers to its application in a specific training environment. We list the environments which different algorithms adapt to in Table 5. AsyLPG is an asynchronous compression algorithm. The training performance in dynamic and heterogeneous resource environments is not dragged down by stragglers. In dynamic networks, DC2 adaptively adjusts the compression ratio in response to the change of bandwidth. It aims to keep the average communication delay close to historical minimum. LAGS-SGD and OMGS-SGD maximize the overlap between communication and computation in the pipeline training paradigm. CE-FedAvg is used in the context of IoT edge computing where system resources are limited and heterogeneous across computing nodes. FL-LSGD-DB is designed to reduce the total consumption of computational and communication resources in wireless federated learning. CAT adopts different compression ratios for different tasks in IoT devices, and adjusts the ratios at different training stages. Liu's method allows model training in wireless federated learning to reach a convergence target within a specified time by adjusting the computation and communication time.

Table 5
The environment which different environment dependent algorithms are adapted to.

Algorithm	Suitable environment or application
AsyLPG	Dynamic or heterogeneous resource environment.
DC2	Network with dynamic bandwidth.
LAGS-SGD	Pipelined training.
OMGS-SGD	Pipelined training.
CE-FedAvg	IoT edge-computing scenario with heterogeneous resources.
FL-LSGD-DB	Wireless network for saving energy consumption.
CAT	Energy-limited IoT devices.
Liu's method	Wireless federated learning to reduce the learning latency.

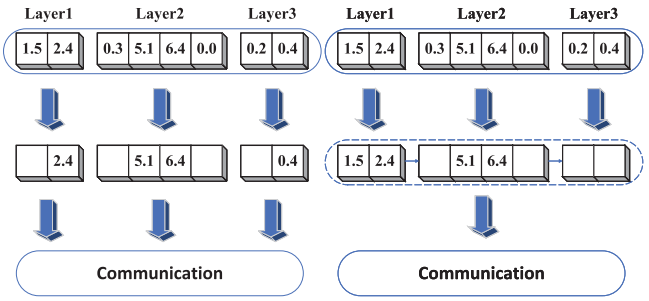
8. Comparison of different compression methods

The communication compression methods discussed in the previous sections are compared in five aspects. The first is the scope of compression that scrutinizes the compression of per-layer gradients and these of the whole model. The second is the location of compression in parameter server architecture including the one-way and bidirectional compressions between the workers and the servers. The third is the performance comparison of versatile compression methods for non-i.i.d data. The fourth is the analysis of the compression/decompression overhead in practical deployment. The fifth is the comparison of theoretical convergence rate for different objective functions. In addition, we present and analyze the advantages and disadvantages of each compression method. In the end of this section, we propose several potential research directions in communication compression.

8.1. Layer-wise compression and global compression

The scope of the compression determines whether to layer-wise or globally compress the gradient elements. Layer-wise compression means that the compression is performed separately for each layer of a model. Global compression first concatenates the tensors of all layers to form a new gradient tensor containing all elements, and then performs the compression on this new tensor.

In practical scenarios, the compression methods are usually applied layer-by-layer. Layer-wise compression is efficient and easy to implement in the commonly used distributed deep learning toolkits, such as PyTorch and TensorFlow. In these toolkits, the gradients of the model are generated layer-by-layer in the backpropagation process. Once the gradient tensor of each layer is generated, the compression can be performed immediately [133]. Layer-wise compression can also increase the overlap between the computation and the communication to further reduce the training time [125]. Global compression needs to wait until the full backpropagation is finished and all the gradient tensors of the whole model are obtained. The time to wait for the entire backpropagation and concatenating all layer gradient tensors is not negligible. In addition, it has been demonstrated through theoretical derivation that the compression error generated in layer-wise compression is strictly smaller than that in global compression [31].



(a) Layer-wise Sparsification (b) Global Sparsification

Fig. 7. Layer-wise sparsification and global sparsification.

In the communication sparsification, only a small fraction of the original elements is updated at each iteration. Layer-wise sparsification can guarantee that the per-layer parameters of the model can be updated. However, global sparsification may lost information from some layers, as is shown in Fig. 7. In most unbiased uniform quantization methods like QSGD and TernGrad, layer-wise quantization can obtain higher test accuracy [31]. The calculation of scaling factor in these quantization is significantly affected by the extreme element of the original tensor [61,63]. Single scaling factor from entire model may be inappropriate in many layers. Most elements will be quantized to the values with large deviations. Thus, global quantization has higher probability of error than layer-wise quantization. In contrast, layer-wise quantization can effectively alleviate this problem. The scaling factor is selected separately in different layers and the distribution of quantized element is balanced.

In the vast majority of cases, layer-wise compression performs better than global compression [23,31]. Layer-wise compression is more straightforward in the modern distributed deep learning toolkits and can achieve a more balanced compression result. In addition, the overlap between computation time and communication time can be increase to shorten single iteration time. However, the startup time problem of layer-wise communication in deep models may result in worse performance [73,126].

8.2. One-way compression and bidirectional compression

In the parameter server architecture, most communication compression methods only improve the uplink communication. There are some methods that focus on how to compress both uplink and downlink communications. DoubleSqueeze [134] deploys the error accumulation compensation in the workers and the servers respectively to perform the bidirectional communication compression.

In the communication quantization, NC [84] and TernGrad [63] both perform the compression at the worker side and the server side separately. Extreme quantitation method SignSGD [57] uses the majority voting mechanism to complete the compression of downlink communication. The aggregation result of all one-bit messages in the server remains one bit. It is the same as the sign value of more than half of the workers. AsyLPG [120] quantizes the gradients in the uplink communication and quantizes the parameters in the downlink communication. In the communication sparsification, there will be a "gradient built-up" problem in the server when the workers select different elements for communication [28,46,51]. The huge aggregated result may make the downlink communication congested. This problem will become more serious when the number of workers increases [29]. Even in a very large distributed system, the downlink communication volume may be the same as that in the uncompressed case. To solve this problem, ScaleCom [29] and Sketched-SGD [45] unify the indices of filtered elements in all the workers. The gTop-k [28] performs Top-k sparsification again on the server to reduce the downlink communication volume.

Table 6
The robustness to non-i.i.d data in different compression algorithms.

Algorithm	Robust to non-i.i.d data	Category
TernGrad	No	Quantization
QSGD	No	Quantization
ATOMO	No	Quantization
SignSGD	No	Quantization
Top-K	Yes	Sparsification
GradDrop	Yes	Sparsification
DGC	Yes	Sparsification
EGC	Yes	Sparsification
SBC	Yes	Hybrid
FedZip	Yes	Hybrid
Variance-based	Yes	Hybrid

8.3. Robustness to non-i.i.d data

In traditional distributed learning, the data are distributed on a cluster and sampled independently from the same distribution, i.e., the data are independently and identically distributed (i.i.d). However, in some cases, the computing devices belong to different users and enterprises. The distribution of data is non-i.i.d across devices, and data partition sizes are unbalanced. The non-i.i.d and unbalanced data bring challenges to the performance of communication compression. Non-i.i.d data is one of the main challenges faced by decentralized learning or federated learning in the real world, which will lead to deterioration of learning performance. The communication acceleration of compression comes at the cost of lossy exchange of gradients or parameters, which probably slows down the convergence speed and impairs model accuracy. Communication compression algorithms designed without considering the influence of non-i.i.d data cannot work well, resulting in inferior model accuracy. It is necessary and beneficial to explore the performance of different compression methods in non-i.i.d data and provide instructions on how to select the suitable compression method.

Many decentralized compression methods are specifically designed for the non-i.i.d data. Sparse-Push [110] proposes Skew-Compensated Sparse-Push based on itself. All the workers train the model with Sparse-Push for the first vast majority of iterations. For the subsequent iterations, the workers perform the Push-sum algorithm without compression for correcting the optimization direction. SkewScout [135] periodically moves the model from one worker to another for evaluating the accuracy of the model on different workers. It can infer the accuracy loss of local model on different data partitions. SkewScout builds an optimization problem that minimizes the communication overhead given the maximum accuracy loss. An adaptive compression ratio control is performed by solving this optimization problem.

Table 6 lists the robustness of many common distributed compression methods to non-i.i.d data. All compression methods suffer from a decrease in convergence rate, but the sparsification based on magnitude is least affected. The robustness to non-i.i.d data of quantization methods TernGrad, QSGD, ATOMO and SignSGD is not good [136]. In SignSGD, highly non-i.i.d data causes the model update weakly correlated with the steepest descent direction. In the extreme case where each worker only holds data from a single class, SignSGD may fail to converge at all. The sparsification methods exhibit very much better performance than quantization methods in non-i.i.d data. There is no distinct slowdown in the convergence rate in the extreme non-i.i.d data. The Top-k sparsification can ensure the training stability because the compression noise is not amplified by quantization like SignSGD [136]. Besides, the transmission of important elements in sparsification keeps the deviation among the workers from being too far. Therefore, the sparsification methods based on magnitude is not affected by weight divergence and robust to non-i.i.d data.

8.4. Compression and decompression overhead

Despite the remarkable decrease in the transmission time, the communication compression methods also introduce additional computational overheads during the compression and decompression processes. For example, the bidirectional compression requires performing the compression and decompression in the both sides of parameter server architecture. Moreover, in the All-Gather architecture, each worker decompresses the gradients multiple times in a single iteration. Under such circumstances, the additional computational overhead is significant. Consequently, excessive compression overhead may negate or even counteract the desired effects of reducing the training time, especially when the bandwidth is sufficiently high [137,138]. A lightweight compression method in computational overhead is necessary, especially in the scenarios where computing resources are limited. In addition, the design of compression method should consider the compatibility with GPU computation since it also has some impact on the computational overhead [35].

The computational overhead of communication sparsification methods is widely discussed. The original Top-k selects the elements by a sorting algorithm, which has a high computational complexity. In Pytorch toolkits, the efficient Top-k selection is the RadixSelect algorithm [139]. RadixSelect determines the bit element of the kth largest element with scan and scatter operations. It has a computational complexity of $O(\lceil b/r \rceil n)$. Here, b is the bit width and r is the radix size. However, the scan operation and the scatter operation are not suitable for parallel processing and are very time-consuming in GPU [35]. Due to the irregular memory access, the exact Top-k selection is not friendly to the GPU computation [140,141]. Therefore, the original Top-k is not suitable for large-scale GPU clusters.

Threshold comparison becomes the mainstream sparsification for its low complexity of filtering gradients. However, the selection of suitable threshold may also incur the huge computational overhead. In order to reduce the overhead, many solutions have been proposed. GradDrop [34] and DGC [32] estimate the threshold by random sampling and sorting the small samples. HGC [92] makes a reasonable threshold estimation by the mean value of the elements. Both RedSync [35] and MSTop-k [36] obtain a threshold by binary search. They are friendly to GPU computation because the memory access is regular. Although threshold comparison can effectively reduce the complexity, the filtered results may be less accurate [41]. The adjustment of the obtained results might make the compression time much long. There are various methods to select important elements. GradSparse [24] gives each element a probability of being selected based on its absolute value, and a greedy algorithm is designed for improving computational efficiency. ProbComp-LPAC [25] uses a probability equation to determine whether an element is in the Top-k range. In Variance-based sparsification [94], the gradients are selected according to their variance. Computing the variances is time-consuming, so it achieves the effect of comparing the mean absolute value and the variance by comparing the squared mean and the sum of squares.

Common quantization consists of scaling and mapping. The calculation of suitable scaling factor is inseparable from the traversal of the overall elements. The unbiased mapping for each element usually relies on random processes. The time to generate random variables for large number of elements mapping is non-negligible [71]. The random quantization methods, such as QSGD and TernGrad, have a complexity order of $O(n^2)$ [92]. In addition, the overhead of the quantization is more sensitive to the compression ratio than the sparsification [142].

The main computational overhead of quantization lies in floating-point operations. The computation of NC is equivalent to removing the mantissa part in the binary representation of the value. It does not need time-consuming floating-point addition and multiplication operations. APoT [85] also avoids large floating-point operations in the compression or decompression processes and uses only the more efficient bit operations (left-shift or right-shift operations). For the dynamic quantization methods, the computational overhead in the auto-decision of quantization levels is also non-negligible [78]. MQGrad [80] uses a reinforcement learning-based approach to increase quantization levels at each training iteration. Along the training iterations, Oland et al. [79] propose to dynamically select quantization levels using the root-mean-square values of the gradients at each level. Both of these dynamic quantization methods are accompanied by a large computational overhead. Although AdaQS [77] selects the degree of quantization by variance, it uses an approximate variance estimation algorithm. The extra overhead of AdaQS is largely optimized compared with MQGrad and Oland's method.

Table 7
Theoretical convergence rates of different compression algorithms.

Algorithm	Convergence rate		
	Strongly convex	Convex	Non-convex
Top-k	$\mathcal{O}(1/T)$	–	$\mathcal{O}(1/\sqrt{T})$
Random-k	$\mathcal{O}(1/T)$	–	$\mathcal{O}(1/\sqrt{T})$
FedPAQ	$\mathcal{O}(1/T)$	–	$\mathcal{O}(1/\sqrt{T})$
DIANA	$\mathcal{O}(1/T)$	–	$\mathcal{O}(1/\sqrt{T})$
Qsparse	$\mathcal{O}(1/(nT))$	–	$\mathcal{O}(1/\sqrt{nT})$
SIDCo	–	–	$\mathcal{O}(1/\sqrt{T})$
QSGD	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
TernGrad	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
EFSignSGD	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
ALQ/AMQ	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
NUQSGD	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
ScaleCom	–	–	$\mathcal{O}(1/\sqrt{nT})$
EGC	–	–	$\mathcal{O}(1/\sqrt{T})$
rTop-k	–	–	$\mathcal{O}(1/\sqrt{nT})$
SignSGD	–	–	$\mathcal{O}(1/\sqrt{T})$
NC	–	–	$\mathcal{O}(1/\sqrt{nT})$
MARINA	–	–	$\mathcal{O}(1/\sqrt{T})$
1-bit Adam	–	–	$\mathcal{O}(1/\sqrt{nT})$
ECX in SGD	–	–	$\mathcal{O}(1/\sqrt{nT})$
ECX in STORM	–	–	$\mathcal{O}(1/(nT)^{\frac{2}{3}})$
ECX in IGT	–	–	$\mathcal{O}(1/(nT)^{\frac{4}{9}})$

T: Training rounds.
n: The number of workers.

8.5. Theoretical convergence rate

Communication compression effectively reduces the wall-clock time of each iteration, while prolonging the number of training rounds till convergence. The underlying reason is that the compressed gradients may deviate from the true gradients, and their variances are amplified with random sampling of data during training. The higher compression degree, the larger the deviation and the variance. The exorbitant gradient compression will cause a very long convergence time and poor model performance. Hence, it is crucial to understand how the convergence rate is impaired by gradient compression.

In theory, the convergence analysis demands appropriate assumptions, especially the convexity of the loss function. It measures the curvature in the relationship between values of loss function and model parameters. The subtle differences in the assumption may lead to distinct convergence rates. Depending on the assumptions, we categorize the known results in three groups: *Strongly Convex*, *Convex* and *Non-convex*. We hereby clarify that SGD serves as the baseline algorithm. For the strongly convex loss function, we focus on SGD with a decreasing learning rate. The convergence rate is $\mathcal{O}(1/T)$, where T is the number of iterations. For a convex or non-convex loss function, SGD with a

Table 8
Theoretical convergence rates with different compression frameworks.

Compression framework	Convergence rate			Compression algorithms	
	Strongly convex	Convex	non-convex	Unbiased compression	Biased compression
Choco-SGD	$\mathcal{O}(1/(nT))$	–	$\mathcal{O}(1/\sqrt{nT})$	✓	✓
ADC-DGD	–	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$	✓	
Quantized Push-Sum	–	$\mathcal{O}(1/\sqrt{nT})$	$\mathcal{O}(1/\sqrt{nT})$	✓	
swarmSGD	–	–	$\mathcal{O}(1/\sqrt{T})$	✓	
DeepSqueeze	–	–	$\mathcal{O}(1/\sqrt{nT})$	✓	✓
DCD-PSGD	–	–	$\mathcal{O}(1/\sqrt{nT})$	✓	
ECD-PSGD	–	–	$\mathcal{O}(1/\sqrt{nT})$	✓	
Moniqua	–	–	$\mathcal{O}(1/\sqrt{nT})$	Mod compressor	
DeLi-CoCo	Linearity	–	Linearity	✓	✓
LEAD	Linearity	–	–	✓	
C-GT	Linearity	–	–	Special compressor	

T: Training rounds.
n: The number of workers.

constant learning rate is followed with interest. Due to the noise of the stochastic gradient, it achieves a sublinear convergence rate in the form of $\mathcal{O}(1/\sqrt{T})$.

There are many compression algorithms have been proposed in recent years, each with subtle differences in their assumptions and implementations. In this subsection, a shorthand analysis of these algorithms is given in terms of the convergence rate for different objective functions. Table 7 lists the convergence rate of common compression algorithms. Some algorithms with a strongly convex loss function and a decreasing learning rate can converges the same as the baseline SGD. Many algorithms focus on the convergence with non-convex loss functions, and almost all of they converge at rate $\mathcal{O}(1/\sqrt{T})$, the same rate as vanilla SGD. In practical application scenarios, the loss function is usually non-convex, such as neural networks in deep learning. Communication compression is commonly used in the distributed training of deep neural networks due to their large amount of parameters. Therefore, focusing on the convergence rate of the compression algorithm with a non-convex loss function is expected.

The compression frameworks can generally be applied to many compression algorithms. These algorithms are similar in their theoretical representation as an operator in framework, although they differ in design and principle. They can be divided into two types of compression algorithms: unbiased compression and biased compression. The convergence rates of different compression frameworks are listed in Table 8. The types of applicable compression algorithms are also involved. These frameworks with suitable algorithms can achieve the same convergence rate as SGD in order for some objective functions.

Compression methods enhanced by acceleration algorithms or optimization techniques can achieve a faster theoretical convergence rate than vanilla SGD under different objective functions. For example, the theoretical convergence rate of ADIANA and CANITA combined with the acceleration algorithm is superior to the original compression algorithm DIANA under strongly-convex objective functions. With a non-convex loss function, ErrorcompenstatedX can only achieve a convergence rate of $\mathcal{O}(1/\sqrt{T})$ using SGD and Momentum SGD. If boosted by stochastic recursive momentum and implicit gradient transport, ErrorCompenstatedX (ECX) is able to achieve the convergence rates of $\mathcal{O}(1/(nT)^{\frac{2}{3}})$ and $\mathcal{O}(1/(nT)^{\frac{4}{9}})$, respectively. Optimization algorithms, including multiple gossip (or multi–consensus) and gradient tracking, are designed for accelerating the convergence rate in a decentralized network. Under strongly convex objectives, DeLi-CoCo, C-GT and LEAD can achieve linear convergence rate, and DeLi-CoCo holds the same convergence rate even under non-convex objective.

It is also necessary to discuss the effect of error accumulation compensation on convergence. Without error compensation, the unbiased compression can achieve the same convergence rate as SGD [143] and the biased compression can also achieve convergence in theory [31,144]. It is noteworthy that the error compensation strategy can partially alleviate the convergence issues of any compression algorithms [58] and improve the model accuracy. In general, the higher the degree of compression, the slower the convergence rate. In terms of theoretical convergence rate, the impact of compression ratio lies in the high order term after deploying error accumulation compensation. The dominant term of convergence rate is generally consistent with that of the uncompressed case. In terms of actual convergence time, the higher the degree of compression, the longer the training time of the model reaches to convergence. This phenomenon is more pronounced in aggressive compression ratios. However, error accumulation compensation can introduce large noise in large batch training in practice [29,121], increasing the number of iterations needed to converge.

8.6. Advantages and disadvantages

In this section, we analyze and compare the advantages and disadvantages of different compression methods, as shown in Table 9. The advantages of compression methods are summarized in seven aspects: wide range of compression levels, deployability in various communication architectures (Parameter Server and Ring All-Reduce or decentralized training), light compression overhead, robustness to non-i.i.d data training, adaptability to dynamic or heterogeneous networks, tolerance to user loss or attacks, and bidirectional compression.

The wide range of compression levels, i.e., a wide range of compression ratios, provides flexibility in adapting compression to different resource environments, thereby meeting diverse training requirements. In different network environments, the compression level can be adjusted according to the network bandwidth to improve the bandwidth utilization while accelerating the training. What is more, compression algorithms with the advantage of wide compression levels can be deployed in applications where the communication volume is precisely required. They can be used for maximizing the computation-communication overlap in pipeline training or the combination of compression and other scheduling techniques. Most of the sparsification and hybrid compressions have a large selection space of compression levels. The compression level of most quantization algorithms is discrete and does not exceed 32 times, because each element requires at least 1 bit as its representation. With the help of some auxiliary techniques (e.g., error accumulation compensation) that help to ensure convergence, compression algorithms are usually able to achieve higher compression degree than the original. Model accuracy is generally related to the compression ratio deployed by the algorithm [145]. A significant decrease in model accuracy may occur

Table 9
Advantages and disadvantages of different compression methods.

Advantages \ Disadvantages	Limited compression levels	Additional memory required	Heavy compression overhead	No theoretical guarantee	Unfriendly hyperparameters
Wide compression levels		Random-k [22] Top-k [22] rTop-k [27] gTop-k [28] DGC [32] GradDrop [34] EGC [37] GradSA [39] SDAGC [40] Gaussian-k [41] SIDCo [42] 3LC [71] Qsparse [88] SBC [91] HGC [92] AdaComp [95]	Top-k [22] rTop-k [27] gTop-k [28] GradSA [39] Qsparse [88] Varianced [94] AdaComp [95]	LGC [30] GradDrop [34] 3LC [71] Sparse 1-bit [89] FEDZIP [93] AdaComp [95] Varianced [94] AdaComp [95] MGC [96] DeepReduce [97,98]	Sparse 1-bit [89] FEDZIP [93] MGC [96]
Suitable for various architectures	Sketched-SGD [45] FetchSGD [46] GradiVeQ [50] PowerSGD [51] FP16 [53] 8-BIT [54] SignSGD [56,57] ATOMO [76] UVeQFed [82] DCD-PSGD [101] DC-DGD [104] Moniqua [106] GossipFL [107] QPush-sum [109]	Scalecom [29] EGC [37] Choco-SGD [102] SwarmSGD [108] Sparse-Push [110]	GradiVeQ [50] PowerSGD [51] ATOMO [76] UVeQFed [82]	LGC [30] MSTop-k [36] GradiVeQ [50] PowerSGD [51] FP16 [53] 8-BIT [54]	
Light compression overhead	FP16 [53] 1-Bit SGD [55] SignSGD [56,57] LE-SignGD [59] 1-bit Adam [60] QSGD [61] ECQ-SGD [62] TernGrad [63] CD-SGD [73] FedPAQ [18] PQASGD [74] NC [84] APoT [85] SketchML [99,100] Moniqua [106] QPush-sum [109] AsyLPG [120]	DGC [32] GradDrop [34] Gaussian-k [41] SIDCo [42] 1-Bit SGD [55] EFSignSGD [58] 1-bit Adam [60] ECQ-SGD [62] AsyLPG [120]		Probcomp-LPAC [25] LGC [30] GradDrop [34] RedSync [35] MSTop-k [36] FP16 [53] 1-Bit SGD [55] AdaQS [77,78] APoT [85] DeepReduce [97,98]	Probcomp-LPAC [25] Threshold-v[31] FedPAQ [18] PQASGD [74]
Robust to non-i.i.d data	Sketched-SGD [45] FetchSGD [46]	Top-k [22] gTop-k [28] DGC [32] GradDrop [34] EGC [37] SBC [91] SwarmSGD [108] Sparse-Push [110]	Top-k [22] gTop-k [28] Varianced [94]	GradDrop [34] FEDZIP [93] Varianced [94]	FEDZIP [93]
Adaptable to dynamic or heterogeneous network	MARINA [70] Qsparse [88] GossipFL [107] AsyLPG [120]	MARINA [70] SwarmSGD [108] Sparse-Push [110] AsyLPG [120]	Qsparse [88] LAGS-SGD [125] OMGS-SGD [126]	DC2 [121]	DC2 [121]
Tolerate lost or attacked workers	FetchSGD [46] SignSGD [56,57] MARINA [70] FedPC [72] FedPAQ [18] AQG [81]	EGC [37] MARINA [70] FedPC [72] AQG [81] Sparse-Push [110]	AQG [81]	FedPC [72]	FedPAQ [18]
Bidirectional compression	TernGrad [63] DIANA [64] MARINA [70] NC [84] AsyLPG [120] AsyLPG [120]	gTop-k [28] Scalecom [29] DIANA [64] MARINA [70] SBC [91]	gTop-k [28]		

when the degree of compression is higher than a certain value. Some literature suggests a logarithmic relationship between the compression ratio and the accuracy of the compressed model [146]. In practical experiments, sometimes a high degree of compression can help to improve the generalization ability of the model, because the noise generated by compression may help the model to jump out of the local optimum [61].

The deployability in different communication architectures determines the application scope of the compression algorithms. The popular architectures for distributed deep learning include All-Reduce and decentralized communication topologies, in addition to the parameter server. In data centers, Ring All-Reduce has become a widely used communication architecture due to its low communication latency, which is not correlated with the number of workers involved. However, it is difficult to directly deploy general compression algorithms in Ring All-Reduce while remaining the desired training speedup. The main reason is that compressed gradients cannot be aggregated directly prior to decompression. For general sparsification algorithms, two sparsified results cannot be directly summed together due to the difference in the indices selected by each worker. For general quantization algorithms, the cumulative result of two quantization levels may overflow the limited range. Even if decompression is performed before aggregation, there are still two concerns for the deployment of general compression methods. On the one hand, large decompression and compression overhead introduced by Ring All-Reduce is prohibitive, requiring multiple summation operations of gradient chunks. On the other hand, as aggregation continues in Ring All-Reduce, the gradient density of general sparsified results with different indices keeps growing, leading to significant communication overhead. Therefore, the scope of compression levels is an important factor to determine whether the algorithm can be deployed in Ring All-Reduce to effectively accelerate the training. Sketch mapping, matrix decomposition as well as quantization based on truncated representations can be directly deployed in Ring All-Reduce, but they do not exploit the sparsity of gradient elements with a limited compression ratio.

The compression overhead is one of the constraints on the speedup performance of compression algorithms. In scenarios where computational power is limited, the significant compression overhead necessitates aggressive compression to effectively reduce the training time for each round. This will also impair the learning convergence, which may consume additional iterations, lowering the overall acceleration effect of the compression algorithm. When the communication time is reduced to the extreme, the factor limiting the speedup improvement is the compression-related overhead in addition to the computation time of forward propagation and backward propagation. The benefits of robustness to Non-i.i.d data and bidirectional compression have been described in previous sections. The resilience to dynamic or heterogeneous networks is also relevant to the deployment of compression algorithm in real-world and this is described in Section 7. Tolerance to lost or attacked workers is of great interest to users in practical training, which indicates whether the algorithm is robust enough to the loss or betrayal of participants. The compression methods with partial participation and security protection together with their corresponding literature are listed in Table 9.

The disadvantages of compression methods are summarized in five aspects: limited compression rate, need for additional memory, heavy compression overhead, no theoretical convergence guarantees and unfriendly hyperparameter setting.

Limited compression levels and heavy compression overhead will restrict the application of compression algorithm. The compression algorithm without a theoretical guarantee makes it difficult for researchers to conduct in-depth analysis. The relevant algorithms are all listed in Table 9. Compression algorithms with auxiliary mechanisms may require additional memory overhead in workers while providing convergence acceleration and guarantee. In the scenarios where storage resources are limited, such as distributed training in mobile devices, such compression methods cannot be deployed properly. To ensure convergence under extreme compression and control compression error, both error accumulation compensation and the use of local auxiliary variables in many decentralized compression frameworks (e.g., CHOCO-SGD) require workers to allocate additional memory space equal to the size of the original model. Memory overhead is also needed in the compression design based on history information of gradient.

Unfavorable hyperparameter setting can significantly influence learning performance. There not exists an one-size-fits-all set of hyperparameters that can make the compression algorithm perform well in the vast majority of applications. The optimal choice of hyperparameters is usually related to the training task and the resource environment. Users may spend a lot of time searching for suitable hyperparameters before training. The setting of positive and negative thresholds in Threshold-v directly affects the compression degree. Inappropriate thresholds may lead to over-compression, making the model convergence slow, or under-compression, which fails to effectively reduce communication traffic. Compression algorithms involve various hyperparameters that need to be chosen by users during training. These hyperparameters include the four adjustment coefficients in Probcomp-LAPC for determining the layer-wise compression ratio, the number of intervals in EGC, the frequency of periodic communication in PQASGD, the correlation coefficients of AIMD in DC2, and the number of clusters in clustering algorithms such as MUSCS and FEDZIP.

8.7. Potential research directions

Through the study of various compression algorithms and frameworks, we discuss and envision the future development of the communication compression for accelerating learning in three directions: system design, communication traffic and communication architecture.

From the system design perspective, the main concern for practical applications is the wall-clock time required for the model to converge under communication compression. Although model compression shortens the communication time of a single iteration, it introduces compression and decompression overheads and increases the number of iterations required for training. It is desirable to jointly consider the reduction of the time cost of a single iteration and the convergence rate when designing a compression algorithm so as to optimize the training time of the entire learning process. The influence of compression on convergence is not easy to measure directly. Even though many existing compression algorithms provide theoretical guarantees of convergence, they are generally an upper bound derived based on unrealistic assumptions. There is a large gap between this loose result and the actual training result, and hence it is not easy to use them for guiding the training time optimization in practice. How to bridge the gap between the theoretical convergence result and the actual training result is an important problem in the design of compression algorithms.

From the communication traffic perspective, it is possible to further improve the learning speed by combining model compression with various techniques that act on the network, such as congestion control. Existing compression is commonly used in conjunction with large batch training and periodic communication, which is a algorithmic-level and source-acting design. These techniques require corresponding hyperparameters (i.e., batch size and communication frequency) to be set before training begins. Testing and fine-tuning are always necessary since there is no universal set of hyperparameters that are effective for different applications and problems. The combination of compression and the network congestion control is a prospective research direction. It can adapt to a variety of environments and applications by compressing or dropping packet of information according to the actual network conditions without any manual adjustment.

From the architecture point of view, the optimization of compression in different communication architectures is important. Currently,

in addition to parameter server architectures, All-Reduce architectures (especially Ring All-Reduce) and decentralized architectures are also very common. In the collective communication, sparsification and quantization are always deployed in the All-Gather architecture, where the communication overhead is highly dependent on the population of workers. The communication latency increases with the number of workers while that of Ring All-Reduce is independent of worker population. All-Gather is not suitable for distributed training with large clusters because the compressed All-Gather communication may take a longer time than that of the uncompressed Ring All-Reduce communication. However, the nature of compression makes it difficult to deploy compression on Ring All-Reduce to achieve a high speedup ratio [50]. How to design a compression algorithm that can be deployed on Ring All-Reduce with the ability to effectively reduce the communication overhead is highly desirable for the distributed training on large-scale clusters. In decentralized training, especially geo-distributed training, the network environment is always dynamic and heterogeneous. Simply setting different compression ratios for different links does not improve training efficiency, as it is likely to violate the model consistency and result in model divergence. It is worthwhile to design suitable communication compression algorithms to improve bandwidth utilization and shorten the training time for decentralized training architectures in dynamic and heterogeneous networks.

9. Conclusion

In this survey, we present a comprehensive survey and taxonomy of the current communication compression methods for distributed deep learning. The communication compression methods are classified into sparsification, quantization, hybrid compression, and context-specific compression according to the method characteristics. For the four compression methods, we further subdivide them based on their specific designs. The algorithm purpose and implementation details of different methods are elaborated. In addition, the performance of different methods in five major aspects are discussed and compared. We conclude by proposing three new research directions in communication compression. This survey analyzes and discusses the communication compression from the perspective of algorithm and system, providing a practical reference and an in-depth research guide for future development.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] L. Deng, D. Yu, et al., Deep learning: methods and applications, Found. Trends® Signal Process. 7 (3–4) (2014) 197–387.

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[3] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, et al., Achieving human parity on automatic Chinese to english news translation, 2018.

[4] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, A. Stolcke, The microsoft 2017 conversational speech recognition system, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2018, pp. 5934–5938.

[5] S. Wang, J. Cao, P. Yu, Deep learning for spatio-temporal data mining: A survey, IEEE Trans. Knowl. Data Eng. (2020).

[6] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M.P. Reyes, M.-L. Shyu, S.-C. Chen, S.S. Iyengar, A survey on deep learning: Algorithms, techniques, and applications, ACM Comput. Surv. 51 (5) (2018) 1–36.

[7] J.D.M.-W.C. Kenton, L.K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL-HLT, 2019, pp. 4171–4186.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

[9] D. Bahdanau, K.H. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations, ICLR 2015, 2015.

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

[11] S. Ouyang, D. Dong, Y. Xu, L. Xiao, Communication optimization strategies for distributed deep neural network training: A survey, J. Parallel Distrib. Comput. 149 (2021) 52–65.

[12] S. Shi, Z. Tang, X. Chu, C. Liu, W. Wang, B. Li, A quantitative survey of communication optimizations in distributed deep learning, IEEE Netw. 35 (3) (2020) 230–237.

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

[14] N.S. Keskar, J. Nocedal, P.T.P. Tang, D. Mudigere, M. Smelyanskiy, On large-batch training for deep learning: Generalization gap and sharp minima, in: 5th International Conference on Learning Representations, ICLR 2017, 2017.

[15] E. Hoffer, I. Hubara, D. Soudry, Train longer, generalize better: closing the generalization gap in large batch training of neural networks, Adv. Neural Inf. Process. Syst. 30 (2017).

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

[17] S. Ma, R. Bassily, M. Belkin, The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 3325–3334.

[18] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, R. Pedarsani, Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 2021–2031.

[19] "Delta-Sigma Modulation", Wikipeia. URL http://en.wikipedia.org/wiki/Delta-sigmamodulation.

[20] H. Tang, Y. Li, J. Liu, M. Yan, ErrorCompensatedX: error compensation for variance reduced algorithms, Adv. Neural Inf. Process. Syst. 34 (2021) 18102–18113.

[21] H. Wang, S. Guo, Z. Qu, R. Li, Z. Liu, Error-compensated sparsification for communication-efficient decentralized training in edge environment, IEEE Trans. Parallel Distrib. Syst. 33 (1) (2021) 14–25.

[22] S.U. Stich, J.-B. Cordonnier, M. Jaggi, Sparsified SGD with memory, Adv. Neural Inf. Process. Syst. 31 (2018).

[23] N.F. Eghlidi, M. Jaggi, Sparse communication for training deep networks, 2020, arXiv preprint arXiv:2009.09271.

[24] J. Wangni, J. Wang, J. Liu, T. Zhang, Gradient sparsification for communication-efficient distributed optimization, Adv. Neural Inf. Process. Syst. 31 (2018).

[25] P. Luo, F.R. Yu, J. Chen, J. Li, V.C. Leung, A novel adaptive gradient compression scheme: Reducing the communication overhead for distributed deep learning in the internet of things, IEEE Internet Things J. 8 (14) (2021) 11476–11486.

[26] H. Wang, J. Chen, X. Wan, H. Tian, J. Xia, G. Zeng, W. Wang, K. Chen, W. Bai, J. Jiang, Domain-specific communication optimization for distributed dnn training, 2020, arXiv preprint arXiv:2008.08445.

[27] L.P. Barnes, H.A. Inan, B. Isik, A. Özgür, rTop-k: A statistical estimation approach to distributed SGD, IEEE J. Select. Areas Inform. Theory 1 (3) (2020) 897–907.

[28] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, 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, IEEE, 2019, pp. 2238–2247.

[29] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen, X. Sun, N. Wang, S. Venkataramani, V.V. Srinivasan, W. Zhang, et al., Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training, Adv. Neural Inf. Process. Syst. 33 (2020) 13551–13563.

[30] L. Abrahamyan, Y. Chen, G. Bekoulis, N. Deligiannis, Learned gradient compression for distributed deep learning, IEEE Trans. Neural Netw. Learn. Syst. (2021).

[31] A. Dutta, E.H. Bergou, A.M. Abdelmoniem, C.-Y. Ho, A.N. Sahu, M. Canini, 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, No. 04, 2020, pp. 3817–3824.

[32] Y. Lin, S. Han, H. Mao, Y. Wang, B. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training, in: International Conference on Learning Representations, 2018.

[33] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, J. Wang, Sparse gradient compression for distributed SGD, in: Database Systems for Advanced Applications: 24th International Conference, DASFAA 2019, Chiang Mai, Thailand, April 22–25, 2019, Proceedings, Part II, Springer, 2019, pp. 139–155.

[34] A. Aji, K. Heafield, Sparse communication for distributed gradient descent, in: EMNLP 2017: Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics (ACL), 2017, pp. 440–445.

[35] J. Fang, H. Fu, G. Yang, C.-J. Hsieh, RedSync: reducing synchronization bandwidth for distributed deep learning training system, J. Parallel Distrib. Comput. 133 (2019) 30–39.

[36] 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, Proc. Mach. Learn. Syst. 3 (2021) 401–412.

[37] D. Xiao, Y. Mei, D. Kuang, M. Chen, B. Guo, W. Wu, EGC: Entropy-based gradient compression for distributed deep learning, Inform. Sci. 548 (2021) 118–134.

[38] H.M. Mahmoud, R. Modarres, R.T. Smythe, Analysis of quickselect: An algorithm for order statistics, RAIRO-Theor. Inf. Appl. 29 (4) (1995) 255–276.

[39] B. Liu, W. Jiang, S. Zhao, H. Jin, B. He, GradSA: Gradient sparsification and accumulation for communication-efficient distributed deep learning, in: International Conference on Green, Pervasive, and Cloud Computing, Springer, 2020, pp. 77–91.

[40] M. Chen, Z. Yan, J. Ren, W. Wu, Standard deviation based adaptive gradient compression for distributed deep learning, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID, IEEE, 2020, pp. 529–538.

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

[42] A.M. Abdelmoniem, A. Elzanaty, M.-S. Alouini, M. Canini, An efficient statistical-based gradient compression technique for distributed training systems, Proc. Mach. Learn. Syst. 3 (2021) 297–322.

[43] C.C. Aggarwal, P.S. Yu, A survey of synopsis construction in data streams, in: Data Streams, Springer, 2007, pp. 169–207.

[44] R. Spring, A. Kyrillidis, V. Mohan, A. Shrivastava, Compressing gradient optimizers via count-sketches, in: International Conference on Machine Learning, PMLR, 2019, pp. 5946–5955.

[45] N. Ivkin, D. Rothchild, E. Ullah, I. Stoica, R. Arora, et al., Communication-efficient distributed SGD with sketching, Adv. Neural Inf. Process. Syst. 32 (2019).

[46] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, R. Arora, Fetchsgd: Communication-efficient federated learning with sketching, in: International Conference on Machine Learning, PMLR, 2020, pp. 8253–8265.

[47] A. Yurtsever, M. Udell, J. Tropp, V. Cevher, Sketchy decisions: Convex low-rank matrix optimization with optimal storage, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1188–1196.

[48] S. Gunasekar, J. Lee, D. Soudry, N. Srebro, Characterizing implicit bias in terms of optimization geometry, in: International Conference on Machine Learning, PMLR, 2018, pp. 1832–1841.

[49] Y. Yoshida, T. Miyato, Spectral norm regularization for improving the generalizability of deep learning, 2017, arXiv preprint arXiv:1705.10941.

[50] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N.S. Kim, A. Schwing, M. Annavaram, S. Avestimehr, Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed cnn training, Adv. Neural Inf. Process. Syst. 31 (2018).

[51] T. Vogels, S.P. Karimireddy, M. Jaggi, PowerSGD: Practical low-rank gradient compression for distributed optimization, Adv. Neural Inf. Process. Syst. 32 (2019).

[52] T. Vogels, S.P. Karimireddy, M. Jaggi, Practical low-rank communication compression in decentralized deep learning, Adv. Neural Inf. Process. Syst. 33 (2020) 14171–14181.

[53] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al., Mixed precision training, 2017, arXiv preprint arXiv:1710.03740.

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

[55] F. Seide, H. Fu, J. Droppo, G. Li, 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, 2014.

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

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

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

[59] X. Deng, T. Sun, F. Liu, D. Li, SignGD with error feedback meets lazily aggregated technique: Communication-efficient algorithms for distributed learning, Tsinghua Sci. Technol. 27 (1) (2021) 174–185.

[60] H. Tang, S. Gan, A.A. Awan, S. Rajbhandari, C. Li, X. Lian, J. Liu, C. Zhang, Y. He, 1-bit adam: Communication efficient large-scale training with adam's convergence speed, in: International Conference on Machine Learning, PMLR, 2021, pp. 10118–10129.

[61] D. Alistarh, D. Grubic, J. Li, R. Tomioka, M. Vojnovic, QSGD: Communication-efficient SGD via gradient quantization and encoding, Adv. Neural Inf. Process. Syst. 30 (2017).

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

[63] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, H. Li, Terngrad: Ternary gradients to reduce communication in distributed deep learning, Adv. Neural Inf. Process. Syst. 30 (2017).

[64] K. Mishchenko, E. Gorbunov, M. Takáč, P. Richtárik, Distributed learning with compressed gradient differences, 2019, arXiv preprint arXiv:1901.09269.

[65] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, Vol. 87, Springer Science & Business Media, 2003.

[66] A. Beck, M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM J. Imaging Sci. 2 (1) (2009) 183–202.

[67] Z. Li, D. Kovalev, X. Qian, P. Richtarik, Acceleration for compressed gradient descent in distributed and federated optimization, in: International Conference on Machine Learning, PMLR, 2020, pp. 5895–5904.

[68] Z. Li, P. Richtárik, CANITA: Faster rates for distributed convex optimization with communication compression, Adv. Neural Inf. Process. Syst. 34 (2021) 13770–13781.

[69] Z. Li, ANITA: An optimal loopless accelerated variance-reduced gradient method, 2021, arXiv preprint arXiv:2103.11333.

[70] E. Gorbunov, K.P. Burlachenko, Z. Li, P. Richtárik, MARINA: Faster non-convex distributed learning with compression, in: International Conference on Machine Learning, PMLR, 2021, pp. 3788–3798.

[71] H. Lim, D.G. Andersen, M. Kaminsky, 3Lc: Lightweight and effective traffic compression for distributed machine learning, Proc. Mach. Learn. Syst. 1 (2019) 53–64.

[72] T.-D. Cao, T. Truong-Huu, H. Tran, K. Tran, A federated deep learning framework for privacy preservation and communication efficiency, J. Syst. Archit. 124 (2022) 102413.

[73] E. Yu, D. Dong, Y. Xu, S. Ouyang, X. Liao, CD-SGD: Distributed stochastic gradient descent with compression and delay compensation, in: 50th International Conference on Parallel Processing, 2021, pp. 1–10.

[74] P. Jiang, G. Agrawal, A linear speedup analysis of distributed deep learning with sparse and quantized communication, Adv. Neural Inf. Process. Syst. 31 (2018).

[75] J. Sun, T. Chen, G. Giannakis, Z. Yang, Communication-efficient distributed learning via lazily aggregated quantized gradients, Adv. Neural Inf. Process. Syst. 32 (2019).

[76] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, S. Wright, Atomo: Communication-efficient learning via atomic sparsification, Adv. Neural Inf. Process. Syst. 31 (2018).

[77] J. Guo, W. Liu, W. Wang, J. Han, R. Li, Y. Lu, S. Hu, Accelerating distributed deep learning by adaptive gradient quantization, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2020, pp. 1603–1607.

[78] J. Guo, S. Hu, W. Wang, C. Yao, J. Han, R. Li, Y. Lu, Tail: an automated and lightweight gradient compression framework for distributed deep learning, in: 2020 57th ACM/IEEE Design Automation Conference, DAC, IEEE, 2020, pp. 1–6.

[79] A. Øland, B. Raj, Reducing communication overhead in distributed learning by an order of magnitude (almost), in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2015, pp. 2219–2223.

[80] G. Cui, J. Xu, W. Zeng, Y. Lan, J. Guo, X. Cheng, Mqgrad: Reinforcement learning of gradient quantization in parameter server, in: Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval, 2018, pp. 83–90.

[81] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, W. Ding, Communication-efficient federated learning with adaptive quantization, ACM Trans. Intell. Syst. Technol. 13 (4) (2022) 1–26.

[82] N. Shlezinger, M. Chen, Y.C. Eldar, H.V. Poor, S. Cui, UVeQFed: Universal vector quantization for federated learning, IEEE Trans. Signal Process. 69 (2020) 500–514.

[83] A. Ramezani-Kebrya, F. Faghri, I. Markov, V. Aksenov, D. Alistarh, D.M. Roy, NUQSGD: Provably communication-efficient data-parallel SGD via nonuniform quantization, J. Mach. Learn. Res. 22 (2021) 114–1.

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

[85] Y. Li, X. Dong, W. Wang, Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks, 2019, arXiv preprint arXiv:1909.13144.

[86] F. Faghri, I. Tabrizian, I. Markov, D. Alistarh, D.M. Roy, A. Ramezani-Kebrya, Adaptive gradient quantization for data-parallel sgd, Adv. Neural Inf. Process. Syst. 33 (2020) 3174–3185.

[87] L. Cui, X. Su, Y. Zhou, Y. Pan, Slashing communication traffic in federated learning by transmitting clustered model updates, IEEE J. Sel. Areas Commun. 39 (8) (2021) 2572–2589.

[88] D. Basu, D. Data, C. Karakus, S. Diggavi, Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations, Adv. Neural Inf. Process. Syst. 32 (2019).

[89] N. Strom, Scalable distributed DNN training using commodity GPU cloud computing, in: Sixteenth Annual Conference of the International Speech Communication Association, 2015.

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

[91] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Sparse binary compression: Towards distributed deep learning with minimal communication, in: 2019 International Joint Conference on Neural Networks, IJCNN, IEEE, 2019, pp. 1–8.

[92] K. Hu, C. Wu, E. Zhu, HGC: Hybrid gradient compression in distributed deep learning, in: International Conference on Artificial Intelligence and Security, Springer, 2021, pp. 15–27.

[93] A. Malekijoo, M.J. Fadaeieslam, H. Malekijou, M. Homayounfar, F. Alizadeh-Shabdiz, R. Rawassizadeh, Fedzip: A compression framework for communication-efficient federated learning, 2021, arXiv preprint arXiv:2102.01593.

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

[95] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, K. Gopalakrishnan, Adacomp: Adaptive residual gradient compression for data-parallel distributed training, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, No. 1, 2018.

[96] Q. Lu, W. Liu, J. Han, J. Guo, Multi-stage gradient compression: Overcoming the communication bottleneck in distributed deep learning, in: International Conference on Neural Information Processing, Springer, 2018, pp. 107–119.

[97] H. Xu, K. Kostopoulou, A. Dutta, X. Li, A. Ntoulas, P. Kalnis, DeepReduce: A sparse-tensor communication framework for federated deep learning, Adv. Neural Inf. Process. Syst. 34 (2021) 21150–21163.

[98] K. Kostopoulou, H. Xu, A. Dutta, X. Li, A. Ntoulas, P. Kalnis, DeepReduce: A sparse-tensor communication framework for distributed deep learning, 2021, arXiv preprint arXiv:2102.03112.

[99] J. Jiang, F. Fu, T. Yang, B. Cui, Sketchml: Accelerating distributed machine learning with data sketches, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1269–1284.

[100] J. Jiang, F. Fu, T. Yang, Y. Shao, B. Cui, SKCompress: compressing sparse and nonuniform gradient in distributed machine learning, VLDB J. 29 (5) (2020) 945–972.

[101] H. Tang, S. Gan, C. Zhang, T. Zhang, J. Liu, Communication compression for decentralized training, Adv. Neural Inf. Process. Syst. 31 (2018).

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

[103] A. Koloskova, T. Lin, S.U. Stich, M. Jaggi, Decentralized deep learning with arbitrary communication compression, in: International Conference on Learning Representations, 2019.

[104] X. Zhang, J. Liu, Z. Zhu, E.S. Bentley, Communication-efficient network-distributed optimization with differential-coded compressors, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 317–326.

[105] X. Zhang, J. Liu, Z. Zhu, E.S. Bentley, Compressed distributed gradient descent: Communication-efficient consensus over networks, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 2431–2439.

[106] Y. Lu, C. De Sa, Moniqua: Modulo quantized communication in decentralized SGD, in: International Conference on Machine Learning, PMLR, 2020, pp. 6415–6425.

[107] Z. Tang, S. Shi, B. Li, X. Chu, Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication, IEEE Trans. Parallel Distrib. Syst. 34 (3) (2022) 909–922.

[108] G. Nadiradze, A. Sabour, P. Davies, S. Li, D. Alistarh, Asynchronous decentralized SGD with quantized and local updates, Adv. Neural Inf. Process. Syst. 34 (2021) 6829–6842.

[109] H. Taheri, A. Mokhtari, H. Hassani, R. Pedarsani, Quantized decentralized stochastic learning over directed graphs, in: International Conference on Machine Learning, PMLR, 2020, pp. 9324–9333.

[110] S.A. Aketi, A. Singh, J. Rabaey, Sparse-push: Communication-& energy-efficient decentralized distributed learning over directed & time-varying graphs with non-IID datasets, 2021, arXiv preprint arXiv:2102.05715.

[111] A. Nedić, A. Olshevsky, Distributed optimization over time-varying directed graphs, IEEE Trans. Automat. Control 60 (3) (2014) 601–615.

[112] A. Hashemi, A. Acharya, R. Das, H. Vikalo, S. Sanghavi, I. Dhillon, On the benefits of multiple gossip steps in communication-constrained decentralized federated learning, IEEE Trans. Parallel Distrib. Syst. 33 (11) (2021) 2727–2739.

[113] Y. Liao, Z. Li, K. Huang, S. Pu, A compressed gradient tracking method for decentralized optimization with linear convergence, IEEE Trans. Automat. Control (2022).

[114] Y. Kajiyama, N. Hayashi, S. Takai, Linear convergence of consensus-based quantized optimization for smooth and strongly convex cost functions, IEEE Trans. Automat. Control 66 (3) (2020) 1254–1261.

[115] X. Liu, Y. Li, Linear convergent decentralized optimization with compression, in: International Conference on Learning Representations, 2021.

[116] Y. Li, X. Liu, J. Tang, M. Yan, K. Yuan, Decentralized composite optimization with compression, 2021, arXiv preprint arXiv:2108.04448.

[117] H. Tang, X. Lian, S. Qiu, L. Yuan, C. Zhang, T. Zhang, J. Liu, {$DeepSqueeze$}: Decentralization meets error-compensated compression, 2019, arXiv preprint arXiv:1907.07346.

[118] B. Liu, Z. Ding, A consensus-based decentralized training algorithm for deep neural networks with communication compression, Neurocomputing 440 (2021) 287–296.

[119] D. Gündüz, P. de Kerret, N.D. Sidiropoulos, D. Gesbert, C.R. Murthy, M. van der Schaar, Machine learning in the air, IEEE J. Sel. Areas Commun. 37 (10) (2019) 2184–2199.

[120] Y. Yu, J. Wu, L. Huang, Double quantization for communication-efficient distributed optimization, Adv. Neural Inf. Process. Syst. 32 (2019).

[121] A.M. Abdelmoniem, M. Canini, DC2: Delay-aware compression control for distributed machine learning, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021, pp. 1–10.

[122] S. Agarwal, H. Wang, K. Lee, S. Venkataraman, D. Papailiopoulos, Adaptive gradient communication via critical learning regime identification, Proc. Mach. Learn. Syst. 3 (2021) 55–80.

[123] S. Jastrzkebski, Z. Kenton, N. Ballas, A. Fischer, Y. Bengio, A. Storkey, On the relation between the sharpest directions of DNN loss and the SGD step length, in: International Conference on Learning Representations.

[124] A. Achille, M. Rovere, S. Soatto, Critical learning periods in deep networks, in: International Conference on Learning Representations.

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

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

[127] S. Shi, X. Chu, B. Li, MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 172–180.

[128] V. Gupta, D. Choudhary, P. Tang, X. Wei, X. Wang, Y. Huang, A. Kejariwal, K. Ramchandran, M.W. Mahoney, Training recommender systems at scale: Communication-efficient model and data parallelism, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2928–2936.

[129] J. Mills, J. Hu, G. Min, Communication-efficient federated learning for wireless edge intelligence in IoT, IEEE Internet Things J. 7 (7) (2019) 5986–5994.

[130] L. Li, D. Shi, R. Hou, H. Li, M. Pan, Z. Han, To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021, pp. 1–10.

[131] S. Khirirat, S. Magnússon, A. Aytekin, M. Johansson, A flexible framework for communication-efficient machine learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, No. 9, 2021, pp. 8101–8109.

[132] S. Liu, G. Yu, R. Yin, J. Yuan, F. Qu, Adaptive batchsize selection and gradient compression for wireless federated learning, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–6.

[133] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, E.P. Xing, Poseidon: An efficient communication architecture for distributed deep learning on {gPU} clusters, in: 2017 USENIX Annual Technical Conference, USENIX ATC 17, 2017, pp. 181–193.

[134] H. Tang, C. Yu, X. Lian, T. Zhang, J. Liu, Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression, in: International Conference on Machine Learning, PMLR, 2019, pp. 6155–6165.

[135] K. Hsieh, A. Phanishayee, O. Mutlu, P. Gibbons, The non-iid data quagmire of decentralized machine learning, in: International Conference on Machine Learning, PMLR, 2020, pp. 4387–4398.

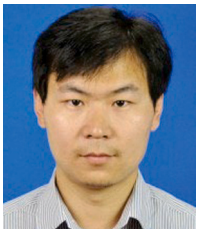
[136] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Robust and communication-efficient federated learning from non-iid data, IEEE Trans. Neural Netw. Learn. Syst. 31 (9) (2019) 3400–3413.

[137] S. Agarwal, H. Wang, S. Venkataraman, D. Papailiopoulos, On the utility of gradient compression in distributed training systems, Proc. Mach. Learn. Syst. 4 (2022) 652–672.

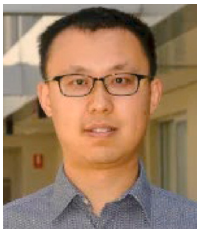
[138] H. Xu, C.-Y. Ho, A.M. Abdelmoniem, A. Dutta, E.H. Bergou, K. Karatsenidis, M. Canini, P. Kalnis, Grace: A compressed communication framework for distributed machine learning, in: 2021 IEEE 41st International Conference on Distributed Computing Systems, ICDCS, IEEE, 2021, pp. 561–572.
[139] A. Sahu, A. Dutta, A. M Abdelmoniem, T. Banerjee, M. Canini, P. Kalnis, Rethinking gradient sparsification as total error minimization, Adv. Neural Inf. Process. Syst. 34 (2021) 8133–8146.
[140] A. Shanbhag, H. Pirk, S. Madden, Efficient top-k query processing on massively parallel hardware, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1557–1570.
[141] X. Mei, X. Chu, Dissecting GPU memory hierarchy through microbenchmarking, IEEE Trans. Parallel Distrib. Syst. 28 (1) (2016) 72–86.
[142] X. Zhang, X. Zhu, J. Wang, H. Yan, H. Chen, W. Bao, Federated learning with adaptive communication compression under dynamic bandwidth and unreliable networks, Inform. Sci. 540 (2020) 242–262.
[143] J. Liu, C. Zhang, et al., Distributed learning systems with first-order methods, Found. Trends® Databases 9 (1) (2020) 1–100.
[144] A. Beznosikov, S. Horváth, P. Richtárik, M. Safaryan, On biased compression for distributed learning, 2020, arXiv preprint arXiv:2002.12410.
[145] S. Salehkalaibar, S. Rini, Lossy gradient compression: How much accuracy can one bit buy? 2022, arXiv preprint arXiv:2202.02812.
[146] P. Li, X. Huang, M. Pan, R. Yu, FedGreen: Federated learning with fine-grained gradient compression for green mobile edge computing, in: 2021 IEEE Global Communications Conference, GLOBECOM, IEEE, 2021, pp. 1–6.

Zeqin Wang received the bachelor's degree from the School of Information Science and Technology, Fudan University in 2021. He is currently pursuing the master's degree with the School of Information Science and Technology, Fudan University. His research interests include distributed machine learning and communication network.

Ming Wen is currently an undergraduate in the school of Information Science and Technology from Fudan University, Shanghai, China. After July 2022, she is going to pursue the master's degree with the School of Information Science and Technology, Fudan University. Her main research interests include distributed machine learning and machine learning theory.



Yuedong Xu received the B.S. degree from Anhui University, the M.S. degree from the Huazhong University of Science and Technology, and the Ph.D. degree from The Chinese University of Hong Kong. From 2009 to 2012, he held a post-doctoral position with INRIA Sophia Antipolis and Universite d'Avignon, France. He is a Professor with the School of Information Science and Technology, Fudan University, China. He has published nearly 20 conference and journal papers in premium vents such as CoNEXT, Mobisys, Mobihoc, Infocom and IEEE/ACM ToN. His research interests include performance evaluation, optimization, machine learning and economic analysis of communication networks and mobile computing.



Yipeng Zhou received the M.Phil. and Ph.D. degrees from Information Engineering (IE) Department, The Chinese University of Hong Kong (CUHK). From 2016 to 2018, he was a Research Fellow with the Institute for Telecommunications Research (ITR), University of South Australia. From 2013 to 2016, he was a Lecturer with the College of Computer Science and Software Engineering, Shenzhen University. He is currently a Lecturer with the Department of Computing, Macquarie University. He is a recipient of the ARC DECRA in 2018.

Jessie Hui Wang received the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2007. Before that, she received the B.S. degree and the M.S. degree in computer science from Tsinghua University. She is currently a tenured Associate Professor with Tsinghua University. Her research interests include Internet routing, distributed computing, network measurement and Internet economics.

Liang Zhang received the Ph.D. degree in circuit and system from Southeast University, Nanjing, China, in 2010. He is currently a Leader Research Engineer with Huawei Technologies Company Ltd. He is currently leading a big data analysis team, focus on the intelligent fault analysis, network health evaluation, and network automation.