Scilight

Edge Intelligence and Systems
https://www.sciltp.com/journals/EIS

Article

Robust Aggregation Oriented Communication Compression via Differential Gradient Sparsification

Zhanbo Jin [1], Haozhao Wang [2,*], Senyao Li [2], Zhigang Zuo [2] and Ruixuan Li [2]

[1] International School, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
* Correspondence: hz_wang@hust.edu.cn

How To Cite: Jin, Z.; Wang, H.; Li, S.; et al. Robust Aggregation Oriented Communication Compression via Differential Gradient Sparsification. *Edge Intelligence and Systems* **2026**, *1*(1), 5.

Received: 19 April 2026
Revised: 3 July 2026
Accepted: 6 July 2026
Published: 23 July 2026

Abstract: Communication efficiency and robustness are two important concerns for distributed learning systems. Gradient sparsification that only transmits partial dimensions of the original gradient is one of the major methods to reduce the communication cost and gradient filter that filters out gradients with abnormal value is the mainstream to tolerate Byzantine attackers. However, in this paper, we identify that existing gradient sparsification techniques are not compatible with the gradient filter because the sparsified gradient may be viewed as abnormal and thus be filtered out. To tackle this challenge, we propose DGSB that sparsifies the differential gradient which is the difference between the gradients in two adjacent iterations instead of the original gradient. The principle is that the server can recover the approximately original gradient of each worker by using their sent sparsified differential gradient, and thus the server can apply the gradient filter over these recovered gradients which are close to the original gradients. Experiments conducted on various training tasks demonstrate that, in the presence of Byzantine workers, DGSB achieves over 30-fold communication compression while maintaining stable test accuracy in the evaluated settings, while the traditional gradient sparsification empowered with Byzantine-resilient methods sometimes cannot converge.

Keywords: distributed learning; sparsification; Byzantine workers; differential gradient

1. Introduction

Stochastic Gradient Descent (SGD)-based distributed machine learning has been recognized as a promising solution to training a large-scale model [1, 2]. Parameter Server (PS) is one of the most popular architectures employed to run distributed SGD [3]. A group of workers compute the local gradients in parallel and then push them to the server. The server aggregates all gradients and updates the global model according to the average of local gradients. With the growing number of workers and the increasingly large size of the training models, communication between the server and workers becomes a prominent bottleneck, especially when the bandwidth resources are limited.

The sparsification technique has been widely adopted for solving the bottleneck by converting high-dimensional vectors into low-dimensional ones, such that the communication data size can be significantly reduced. In the distributed learning system, existing works usually apply the sparsification techniques to the gradient compression [4,5]. Specifically, each worker pushes only several dimensions of the gradient to the server, e.g., k largest elements in Mem-SGD [6], which has shown great advances in solving the communication bottleneck.

While the gradient sparsification has witnessed significant achievements, its applicability highly relies on a critical assumption that all gradients sent by workers are correct. However, this assumption is violated in distributed learning systems with Byzantine workers, where malicious attackers may arbitrarily forge gradients. Although massive robust learning methods have been developed for tolerating Byzantine workers, including selecting correct gradients or getting rid of wrong gradients from all gradients [7–9], they rely on a majority-based assumption that

most of the values of every single dimension are sent by honest workers, which can hardly be satisfied by the existing gradient sparsification methods.

In this paper, we propose the Differential Gradient Sparsification with Byzantine workers tolerance, namely DGSB, that enables the sparsification in the presence of Byzantine workers. The workflow of our method is shown in Figure 1. Instead of sparsifying original gradients, DGSB sparsifies the differential gradients which are the difference between gradients of adjacent iterations. With the sparsified differential gradients from all workers, the server recovers the integral gradients that contain all dimensions based on the cached gradients in previous iteration. Through this way, the values for each single dimension come from all workers and naturally satisfy the condition required by Byzantine-resilient methods if the majority of the workers are honest.

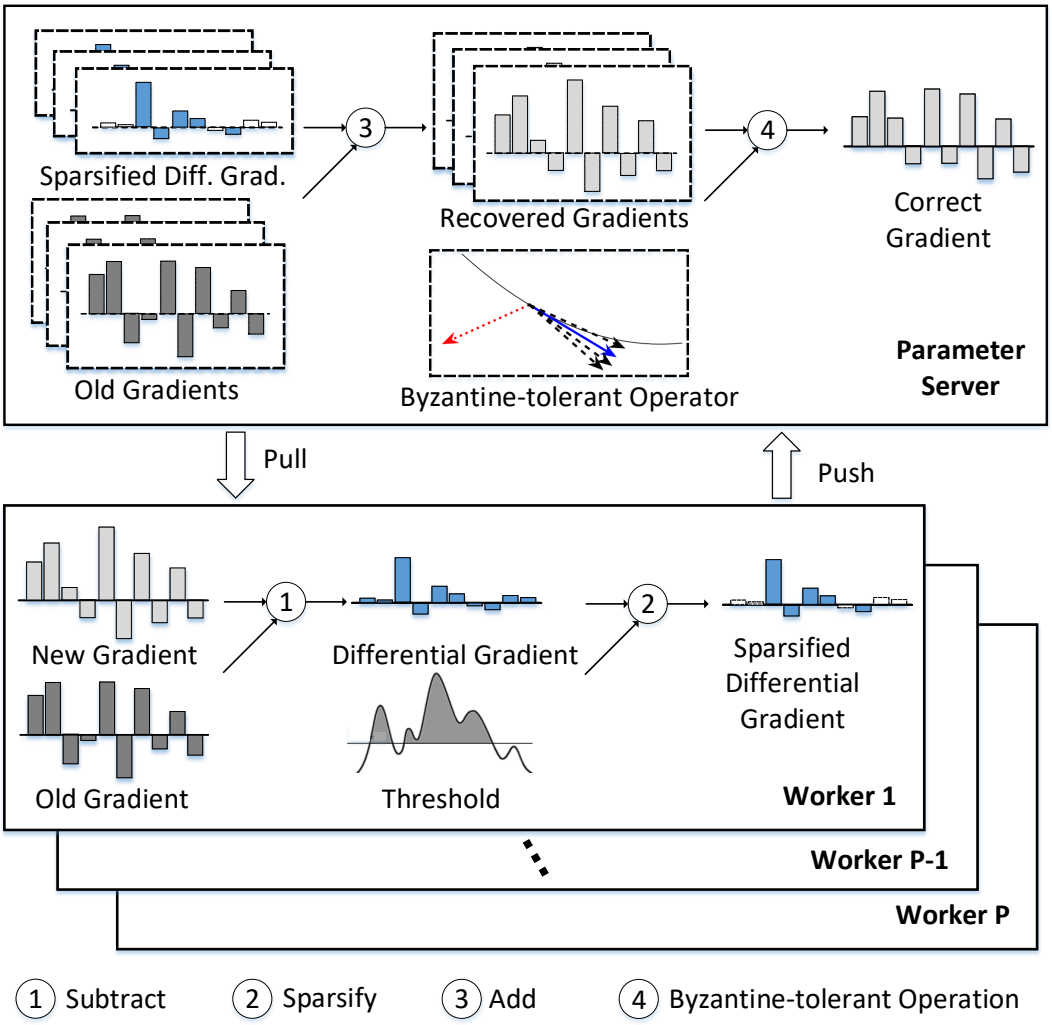


Figure 1. Illustration of the core workflow of DGSB.

Although DGSB can achieve Byzantine tolerance with reduced communication overhead, efficiently sparsifying the differential gradient is quite challenging due to the following two reasons. First, both the convergence rate and the compression efficiency are very sensitive to the selected dimensions of the sparsified differential gradient. Since those dimensions are usually model-dependent, it is hard to determine a suitable sparsification scheme for all models. Another challenge is that the difference between the two gradients of adjacent iterations is usually large, resulting in low compression efficiency. The causes for yielding the large difference mainly comes from two aspects, i.e., the stochasticity(or equivalently variance) of the gradient and the difference of the models for computing the gradients, both of which are basic characteristics of the distributed SGD and difficult to fix without increasing the computing cost.

To tackle these two challenges, in this paper, we further elaborate two technologies. First, to enable sparsifying the differential gradient while retaining convergence efficiency, we design an adaptive threshold to select dimensions based on optimization insights. In practice, the threshold is set to keep compression error under control while preserving training stability. Second, to reduce the difference of the gradients, we propose a heuristic method that

uses the average of the gradients in a sliding window, instead of directly using the stochastic gradient. In fact, such a method is similar to increasing the batch size of SGD leading to the stochasticity reduction. Besides, considering the oscillation characteristic of model updating [10], computing the average of gradients can also naturally mitigate the resulted oscillation of gradients incurred by the model difference and thus reduce the difference of gradients. To verify our proposed algorithm, we further conduct extensive experiments. The results exhibit the great advances of DGSB on compressing the communication of distributed learning in the presence of Byzantine workers.

To be summarized, in this paper, we make the following contributions:

- We propose a novel communication sparsification method called DGSB that is compatible with existing full-gradient Byzantine-resilient methods. Different from traditional gradient sparsification methods, DGSB sparsifies the differential gradient via an adaptive threshold.
- We present two motivation-driven case studies to identify the key bottlenecks of differential gradient sparsification, namely threshold sensitivity and large inter-iteration gradient differences, which guide the design of adaptive thresholding and sliding-window difference reduction.
- We conduct extensive experiments in multiple machine learning tasks. The results show that DGSB achieves over 30-fold communication compression while maintaining stable test accuracy in the evaluated settings with Byzantine workers, while the gradient sparsification based methods sometimes even cannot converge.

This paper is organized as follows. The related work about communication compression and Byzantine tolerance is firstly presented in Section 2. Then, we present the preliminaries and motivations in Section 3. After that, we specify the design of our method DGSB in Section 4. In Section 5, a wide range of evaluations are performed to show the efficiency of DGSB. Finally, the conclusions are drawn in Section 6.

2. Related Work

Recent years have seen an increase in the popularity of tolerating Byzantine workers and sparsifying the communication for the distributed training system.

2.1. Byzantine Tolerance

Byzantine workers are those that push bad gradients, e.g., malicious attacks, to the server such that model training is hindered. Existing methods for tolerating Byzantine workers are to design a robust operator for the gradients aggregation. A foundational work on Byzantine tolerance for synchronous distributed learning was proposed by Blanchard et al. [7], in which they design a method, called Krum, that selects the gradient with the minimum summation of Euclidean distance from the nearest neighbors. After that, from the perspective of robustly estimating the mean of gradients, there are emergingly a number of statistics-based methods, e.g., marginal trimmed mean [11], dimensional median [12,13], and geometric median [8]. To further improve the convergence efficiency of these methods, Xia et al. [9] propose a method called FABA that iteratively removes the outliers from all gradients. Different from the above methods, Chen et al. [14] utilize gradient coding based methods to validate the correctness of gradients through the redundancy of gradient. Targeting at the distributed learning over heterogeneous data, Li et al. [15] normalize the updates sent from workers. Besides, there is also some work aiming at tolerating Byzantine workers of the asynchronous learning [16,17]. Recent studies further address data heterogeneity by adapting robust aggregation through gradient splitting [18]. Recent robust federated learning also explores adaptive clipping strategies to improve resilience under heterogeneous and adversarial settings [19].

2.2. Sparsification

Existing methods for sparsification of distributed learning usually sparsify the gradient sent from workers. The works in early stages for sparsifying gradient used to employ significance-based filters to transmit important updates and suppress the rest [20]. To improve the sparsification efficiency, recently, a large number of works that sparsify the gradient with error compensation have been proposed [4–6,21–23]. Error compensated gradient sparsification is to accumulate the unselect dimensions and compensate the gradient with the accumulated error. In such a way, the convergence gap with non-sparsification distributed SGD could be bridged. Beyond SGD, there are also some work moving towards the Momentum-SGD [24,25]. Different from the methods with error compensation, Wangni et al. [26] propose a novel sparsification operator that the gradient could be sparsified in an unbiased manner. In federated learning, recent communication compression methods further improve update efficiency through learnable binarization, alternative-space lossless sparsification, and enhanced low-rank decomposition [27–29]. Beyond explicit sparsification and quantization, partial network update strategies are also shown to reduce communication and computation overhead in federated learning [30]. More directly, recent work jointly

studies Byzantine robustness and communication compression using variance reduction and error feedback [31]. Though these gradient sparsification based methods have made great achievements in the field of communication compression through sparsification, they are not compatible with current Byzantine-resilient operators. To be summarized, these methods usually tolerate Byzantine workers with a robust operator for gradients aggregation which fail to work when the gradients are sparsified.

3. Background

3.1. Preliminaries

3.1.1. Problem Formulation

Given a dataset $\mathcal{N}$ consisted of n data samples ξ, the machine learning problem is to find the optimal model parameter $\omega \in \mathbb{R}^d$ to minimize the loss of learning the dataset

$$\min_{\omega \in \mathbb{R}^d} F(\omega) = \frac{1}{n} \sum_{i=1}^{n} f(\omega, \xi_i). \tag{1}$$

3.1.2. Distributed SGD in Parameter Server

To solve the optimization problem in Equation (1), the distributed Stochastic Gradient Descent (SGD) in Parameter Server is usually adopted. We denote $\nabla f_i(\omega)$ as the stochastic gradient computed by worker i and N as the number of workers. In each iteration t of distributed SGD, the server updates the parameter ω_t with the average of the received gradients $\mathbf{g}_t^i$ from all workers $i = 1, \ldots, N$:

$$\mathbf{g}_t^i = \nabla f_i(\omega_t), \quad \mathbf{g}_t = Mean(\mathbf{g}_t^1, \ldots, \mathbf{g}_t^N), \quad \omega_{t+1} = \omega_t - \eta \mathbf{g}_t, \tag{2}$$

where η is the learning rate. Since the gradients sent from workers are non-compressed, we call this type of non-compression SGD.

3.1.3. Byzantine Workers Tolerance

Although the distributed SGD update in Equation (2) works efficiently when all workers are honest, it fails to train the model when there are Byzantine workers, as shown in Figure 2. The methods for handling such cases are to employ Byzantine-resilient methods. Assume that among all N workers, P are honest workers and $B = N - P$ are Byzantine workers. General Byzantine-resilient methods apply a robust aggregation operator to all gradients instead of the $Mean(\cdot)$ operator in Equation (2):

$$\mathbf{g}_t = A(\mathbf{g}_t^1, \ldots, \mathbf{g}_t^P, \mathbf{g}_t^{P+1}, \ldots, \mathbf{g}_t^N). \tag{3}$$

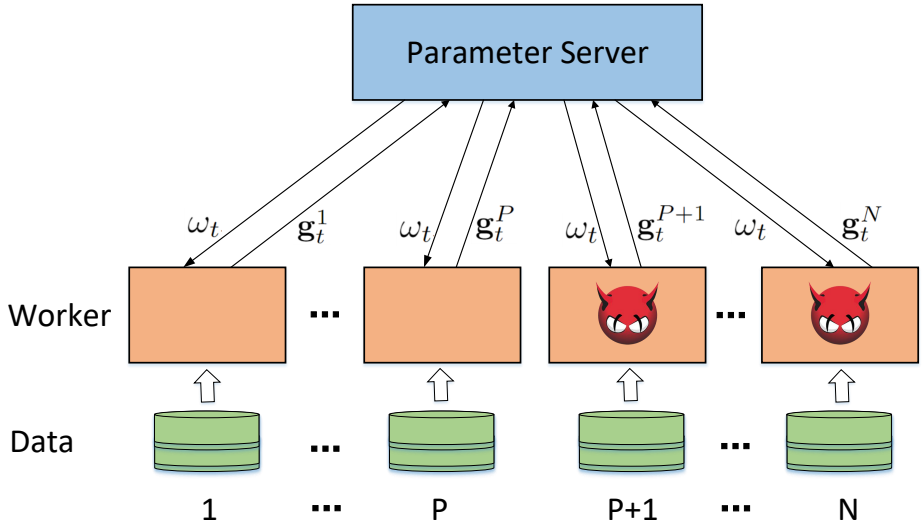


Figure 2. Illustration of distributed SGD in Parameter Server in the presence of Byzantine workers.

The server is assumed to be trusted and to follow the protocol. Honest workers compute and upload updates according to the training algorithm, while Byzantine workers may send arbitrary malicious vectors and may collude with each other. DGSB does not change the fault-tolerance condition of the selected aggregation operator $A(\cdot)$; it is applied under the honest-majority condition required by that operator. In the evaluated Median and FABA settings, this corresponds to $B < N/2$, which is satisfied by our experimental configurations. Note that such a generalization is suitable to all of the existing aggregation operators, as we know, e.g., Krum [7], Trimmed-Mean [11], and FABA [9], which select the correct ones from all gradients based on the majority of the honest workers. Specially, the distributed SGD in (2) can be viewed as a special case when the aggregation operator A is $Mean(\cdot)$.

Before moving on, we introduce the assumptions this paper relying on.

Assumption 1. *We make the following commonly used assumptions:*

(1) ***L-smooth function*** *. The objective function $F(\cdot)$ is L-smooth with L-Lipschitz gradients*

$$\|\nabla F(\omega) - \nabla F(\tilde{\omega})\|_2 \leq L\|\omega - \tilde{\omega}\|_2$$

for all ω, $\tilde{\omega}$.

(2) ***Bounded variance****. The variance of the stochastic gradient is bounded*

$$\mathbb{E}_{\xi}\|\nabla f_i(\omega) - \nabla F(\omega)\|^2 \leq \sigma^2$$

where σ^2 is a constant.

(3) ***Unbiased Gradient****. The stochastic gradient $\nabla f_i(\omega)$ computed from random samples ξ is unbiased for every parameter ω, i.e.,*

$$\mathbb{E}_{\xi}[\nabla f_i(\omega)] = \nabla F(\omega)$$

Note that these assumptions are commonly used in convergence analysis [32,33].

3.2. Motivation

3.2.1. Case Study 1: Sparsification Threshold

We run experiments using the CNN implementation from the official PyTorch MNIST example [34], which is distributed under the BSD 3-Clause License. Figure 3 was generated by the authors from their own experimental results. The experiments are configured with different relative thresholds, i.e., the threshold being set to guarantee a given sparsification ratio. The results show that both the convergence rate and the compression efficiency are very sensitive to the ratio of differential gradient sparsification and achieve different values for the threshold. Besides, the relationship between the convergence rate and the sparsification ratio is non-linear which can be verified by the comparison of the loss reduction from 50% to 10% and from 10% to 5%. Furthermore, the experimental results exhibit that there exists a threshold that achieves the significant communication cost reduction while admitting almost the same convergence rate as non-compressed methods, as exemplified by the sparsification ratio 10% for the CNN. To be summarized, these observations demonstrate the necessity and the possibility of finding a suitable sparsification threshold, and usually, such a sparsification threshold achieves the optimal communication compression when the convergence rate is approximately the same as the non-compression method.

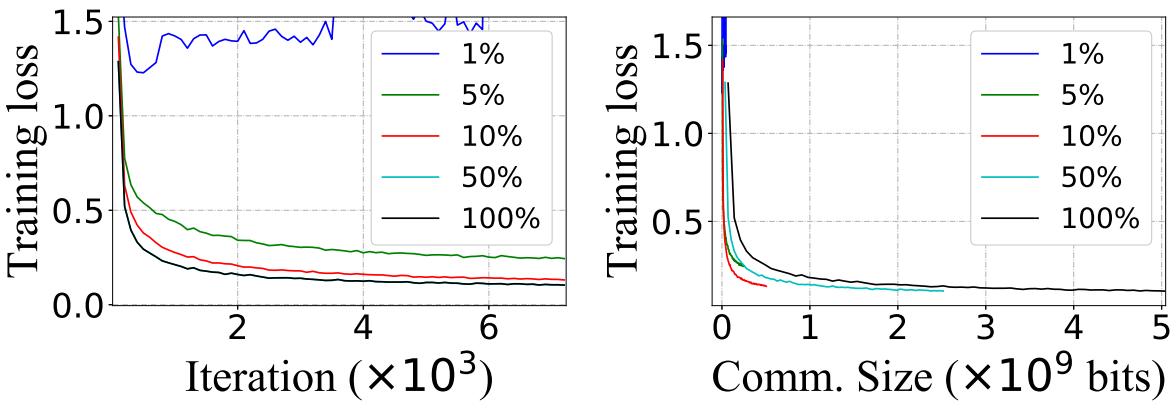


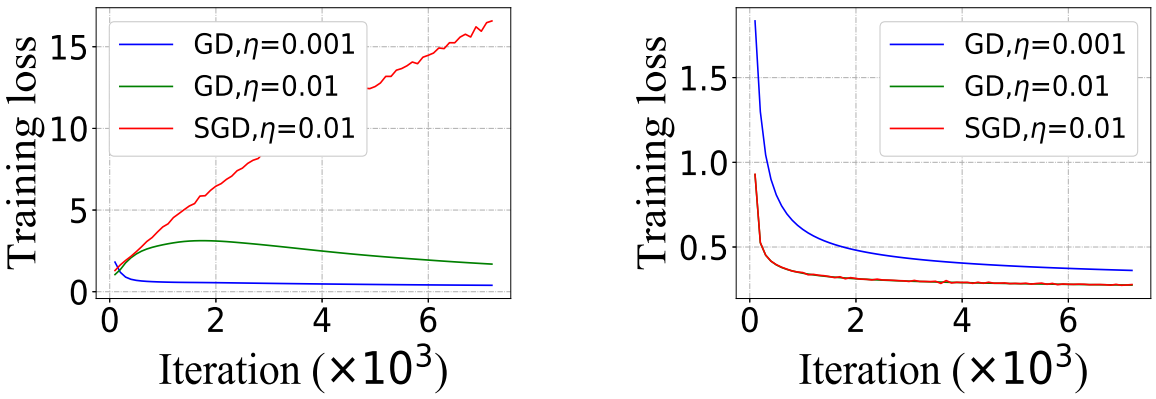
Figure 3. Illustration of the impact of the sparsification ratio.

3.2.2. Case Study 2: Gradient Difference

We run experiments on Logistic Regression (LR) to understand the impact of the difference of the gradients on the compression efficiency. As shown in Figure 4a, the causes of gradient difference reflect on two aspects.

- The gradient variance of SGD incurred by only sampling a random mini-batch of data from the whole dataset has a great impact on the gradient difference. Considering the gradient of Gradient Descent (GD) is computed from the whole dataset that has no variance, we can correspondingly identify the effect of the variance by comparing the SGD and GD under the same conditions with the learning rate being $\eta = 0.01$ and the sparsification ratio being $1/10000$ as shown in Figure 4a. As expected, the results show that the gradient variance has a strongly negative effect on the compression efficiency of the algorithm where the SGD diverges much more severely than the GD although they converge at the same rate when the communication is not compressed as shown in Figure 4b.
- The difference of the models for computing the gradients also has a huge impact on the gradient difference. Since the difference between two models in adjacent iterations is positively related to the learning rate, the impact over the gradient difference can be investigated by comparing GD with the different learning rates, as exemplified by η being 0.001 and 0.01 in Figure 4a. The results that GD with the smaller learning rate converges much better than the GD with the larger learning rate demonstrate a negative relationship between the difference of gradients and the difference of models. In other words, the smaller model difference yields smaller gradient difference allowing higher compression efficiency.

On the other side, the two aspects also indicate the corresponding solutions: increasing batch size for reducing the variance and using small learning rate for reducing the model difference. However, such intuitive methods come with adverse impacts, i.e., increasing the computing cost and reducing the convergence rate. A novel method that can address both challenges is therefore needed.



(**a**) Sparsification (**b**) No Sparsification

Figure 4. Investigation of the causes of the gradient difference: (**a**) sparsification and (**b**) no sparsification.

4. Methodology

In this section, we specify our proposed algorithm. We firstly elaborate the framework of our sparsification algorithm and then specify the methodologies of determining sparsified threshold and reducing gradient difference.

4.1. Differential Gradient Sparsification with Byzantine Tolerance

In this paper, we aim at sparsifying the communication while tolerating Byzantine workers. For tolerating Byzantine workers, the aggregation operators specified in Section 3 has indicated that the whole gradient from each worker is required. Naturally, a sparsification method that enables reconstructing the whole gradient for each worker is necessary, which in fact implies that we could sparsify the gradient difference, namely differential gradient, instead of the gradient itself to enable recovering the gradient. Considering these, we propose a new algorithm DGSB that sparsifies the differential gradient.

The workflow of DGSB is summarized in Algorithm 1. In each iteration t, worker i firstly computes the stochastic gradient $\nabla f_i(\omega_t)$ by using a randomly sampled mini-batch $\xi_{i,t}$ and then reduces its variance with function *difred* to get the stabilized gradient $\mathbf{g}_t^i$. Further, worker i computes the differential gradient Δ_t^i by computing the difference between the newest obtained gradient $\mathbf{g}_t^i$ and the cached gradient $\tilde{\mathbf{g}}_{t-1}^i$ approximating for previous

stabilized gradient $\mathbf{g}_{t-1}^i$:

$$\Delta_t^i = \mathbf{g}_t^i - \tilde{\mathbf{g}}_{t-1}^i. \tag{4}$$

Now, to reduce the communication overhead, worker i elaborately decides a threshold H_t with the function *thresd* and sparsifies the differential gradient Δ_t^i with the threshold by selecting each dimension d of which the magnitude is larger than the threshold

$$\tilde{\Delta}_t^{i,d} = \begin{cases} \Delta_t^{i,d}, & |\Delta_t^{i,d}| > H_t, \\ 0, & |\Delta_t^{i,d}| \leq H_t, \end{cases} \tag{5}$$

and sends the sparsified differential gradient $\tilde{\Delta}_t^i$ to the server.

After receiving the sparsified differential gradient $\tilde{\Delta}_t^i$ from each worker i, the server recovers the gradient $\tilde{\mathbf{g}}_t^i$ with the same cached gradient $\tilde{\mathbf{g}}_{t-1}^i$ as that of the worker to approximate for the gradient $\mathbf{g}_t^i$ of worker i

$$\tilde{\mathbf{g}}_t^i = \tilde{\mathbf{g}}_{t-1}^i + \tilde{\Delta}_t^i. \tag{6}$$

Since the recovered gradient might be sent from Byzantine workers, the server further applies the Byzantine-resilient operators to get the correct gradient

$$\tilde{\mathbf{g}}_t = A(\tilde{\mathbf{g}}_t^1, \ldots, \tilde{\mathbf{g}}_t^N). \tag{7}$$

Algorithm 1 DGSB: Differential Gradient Sparsification with Byzantine Workers Tolerance

Input: Number of workers N, learning rate η
Initialize: ω_1, $\tilde{\mathbf{g}}_0^i \leftarrow 0$, $\tilde{\mathbf{g}}_0 \leftarrow 0$;
for $t = 1$ **to** T **do**
  On worker i=1,…,N:
  Compute the stochastic gradient $\nabla f_i(\omega_t)$ from a randomly sampled mini-batch $\xi_{i,t}$;
  Get $\mathbf{g}_t^i$ by reducing difference of gradient $\nabla f_i(\omega_t)$ with function *difred*;
  Compute differential gradient $\Delta_t^i \leftarrow \mathbf{g}_t^i - \tilde{\mathbf{g}}_{t-1}^i$;
  Compute threshold H_t with function *thresd*;
  Sparsify Δ_t^i into $\tilde{\Delta}_t^i$ with threshold H_t;
  Update local cache $\tilde{\mathbf{g}}_t^i \leftarrow \tilde{\mathbf{g}}_{t-1}^i + \tilde{\Delta}_t^i$;
  Push $\tilde{\Delta}_t^i$ to server;
  On server:
  Recover gradient of each worker $\tilde{\mathbf{g}}_t^i \leftarrow \tilde{\mathbf{g}}_{t-1}^i + \tilde{\Delta}_t^i$ with received $\tilde{\Delta}_t^i$;
  Aggregate all gradients with Byzantine fault-tolerant operator $\tilde{\mathbf{g}}_t \leftarrow A(\tilde{\mathbf{g}}_t^1, \ldots, \tilde{\mathbf{g}}_t^N)$;
  Broadcast $\tilde{\Delta}_t \leftarrow \tilde{\mathbf{g}}_t - \tilde{\mathbf{g}}_{t-1}$ to workers;
  On worker i=1,…,N:
  Pull $\tilde{\Delta}_t$ back from server;
  Recover global gradient $\tilde{\mathbf{g}}_t \leftarrow \tilde{\mathbf{g}}_{t-1} + \tilde{\Delta}_t$;
  Update parameter $\omega_{t+1} \leftarrow \omega_t - \eta\tilde{\mathbf{g}}_t$;
end for

Note that only several dimensions of $\tilde{\mathbf{g}}_t^i$ are different from that of $\tilde{\mathbf{g}}_{t-1}^i$ due to the sparsification of the differential gradient, which implies that only a fraction of dimensions of the aggregated gradient $\tilde{\mathbf{g}}_t$ are different from previous gradient $\tilde{\mathbf{g}}_t$. As a consequence, to reduce the communication overhead sent from server to workers, the server computes the differential gradient of global gradient

$$\tilde{\Delta}_t \leftarrow \tilde{\mathbf{g}}_t - \tilde{\mathbf{g}}_{t-1}, \tag{8}$$

and broadcasts the dimensions with non-zero magnitude to all workers.

Finally, with the received differential gradient $\tilde{\Delta}_t$ from server and the cached gradient $\tilde{\mathbf{g}}_{t-1}$ in local, each worker i recovers the global gradient $\tilde{\mathbf{g}}_t$ and uses it to update the model parameter ω_t

$$\tilde{\mathbf{g}}_t \leftarrow \tilde{\mathbf{g}}_{t-1} + \tilde{\Delta}_t, \qquad \omega_{t+1} = \omega_t - \eta\tilde{\mathbf{g}}_t \tag{9}$$

With the updated model parameter ω_{t+1}, each worker i starts the new iteration which has the same steps as that of the iteration t.

4.2. Adaptive Threshold

To maximize the sparsification efficiency while maintaining the convergence rate, in this section, we present the judiciously designed function *thresd* of determining the threshold H_t. The following analysis focuses on the non-Byzantine case with the aggregation operator being *Mean*, where the variance-reduction step is not included. This setting isolates the optimization effect of DGSB's compression and reconstruction mechanism. In Byzantine settings, DGSB relies on the robustness guarantee of the selected aggregation operator after approximate full-gradient reconstruction, rather than introducing a new Byzantine-resilient aggregation rule.

Since the process of SGD is in an iterative manner, the loss is reduced little by little in each iteration for solving the problem (1). In other words, as the sum of the loss reduction $\mathbb{E}F(\omega_{t+1}) - F(\omega_t)$ from the iteration $t = 1$ to the current iteration $t = T$, the loss reduction $\mathbb{E}F(\omega_{T+1}) - F(\omega_1)$ determines the convergence rate of the algorithm. Therefore, to decide a suitable threshold, we need to bridge the sparsified differential gradient $\tilde{\Delta}_t^i$ and the the expected loss reduction $\mathbb{E}F(\omega_{T+1}) - F(\omega_1)$ in iteration t. Considering that the stochastic gradient $\nabla f_i(\omega_t)$ is used $\mathbf{g}_t^i = \nabla f_i(\omega_t)$, we have the following lemma.

Lemma 1. *With the learning rate $\eta < \frac{1}{L}$, Algorithm 1 has the expected loss reduction as:*

$$\mathbb{E}F(\omega_{T+1}) - F(\omega_1) \leq \frac{\eta}{P}\sum_{t=1}^{T}\sum_{i=1}^{P}\mathbb{E}\Big[-\frac{1}{2}\|\nabla F(\omega_t)\|_2^2 + \frac{1}{2}\|\Delta_t^i - \tilde{\Delta}_t^i\|_2^2\Big] + \frac{L\eta^2\sigma^2 T}{2P}. \tag{10}$$

The proof is provided in Appendix A.

In fact, the Lemma 1 suggests that the sparsified differential gradient $\tilde{\Delta}_t^i$ has to be bounded for the convergence of the algorithm.

Lemma 2. *Under the conditions of the learning rate $\eta \leq \frac{1}{L}$ and the scale factor $\lambda_t < 1$, the Algorithm 1 converges if each dimension of differential gradient $\tilde{\Delta}_t^{i,d}, d = 1, \ldots, D$ satisfies*

$$|\Delta_t^{i,d} - \tilde{\Delta}_t^{i,d}| \leq \lambda_t \frac{\|\nabla F(\omega_t)\|_2}{\sqrt{D}}. \tag{11}$$

The proof is provided in Appendix A.

According to (5), each dimension d of $\Delta_t^i - \tilde{\Delta}_t^i$ is

$$\Delta_t^{i,d} - \tilde{\Delta}_t^{i,d} = \begin{cases} 0, & |\Delta_t^{i,d}| > H_t, \\ \Delta_t^{i,d}, & |\Delta_t^{i,d}| \leq H_t, \end{cases} \tag{12}$$

which indicates $|\Delta_t^{i,d} - \tilde{\Delta}_t^{i,d}| \leq H_t$. Consequently, the condition (11) is satisfied if $H_t \leq \lambda_t \frac{\|\nabla F(\omega_t)\|_2}{\sqrt{D}}$. It can be seen that the Lemma 2 *de-facto* suggests a value of the threshold H_t, i.e., $\lambda_t \frac{\|\nabla F(\omega_t)\|_2}{\sqrt{D}}$. However, $\nabla F(\omega_t)$ is unknown to each worker i in iteration t. For the ease of using in practical system, we here employ the previous aggregated gradient $\tilde{\mathbf{g}}_{t-1}$ to approximate the $\nabla F(\omega_t)$. In other words, the threshold is set to be

$$H_t = \lambda_t \frac{\|\tilde{\mathbf{g}}_{t-1}\|_2}{\sqrt{D}}. \tag{13}$$

Here, λ_t is the effective scale factor used in Lemma 2. In the implementation, the server broadcasts the normalized reference magnitude $q_{t-1} = \|\tilde{\mathbf{g}}_{t-1}\|_2/N$, and each worker computes $H_t = s_t q_{t-1}/\sqrt{D}$ using a runtime scale s_t. Therefore, the implementation is equivalent to (13) with $\lambda_t = s_t/N$. We use s to denote the nominal configured upper bound of s_t; the runtime schedule does not increase s_t above s. Later, we show that such approximation is effective in practice through empirical evaluations.

4.3. Difference Reduction

Although the adaptive threshold could guarantee the high convergence rate of Algorithm 1 in expectation, the difference of the stochastic gradients in two adjacent iterations is usually large such that most dimensions of the

differential gradient are larger than the threshold, leading to low compression efficiency. To obtain high compression efficiency, it is necessary for us to reduce the gradient difference, i.e., designing the function *difred*.

Considering the two motivations discussed in Section 3.2, in this paper, we propose a simple but effective technique to reduce the gradient difference with a little memory cost. Specifically, we employ a sliding window for processing the stochastic gradient in the worker. As shown in Figure 5, in each iteration t, each worker i remains the stochastic gradients $\nabla f_i(\omega_{t-\tau})$ in the sliding window and computes the average of them to obtain the gradient with low difference:

$$\mathbf{g}_t^i = \frac{1}{E}\sum_{\tau=0}^{E-1} \nabla f_i(\omega_{t-\tau}). \tag{14}$$

After the iteration finishing, the sliding window moves to the next gradient and the worker discards the stochastic gradient out of the window from the cache.

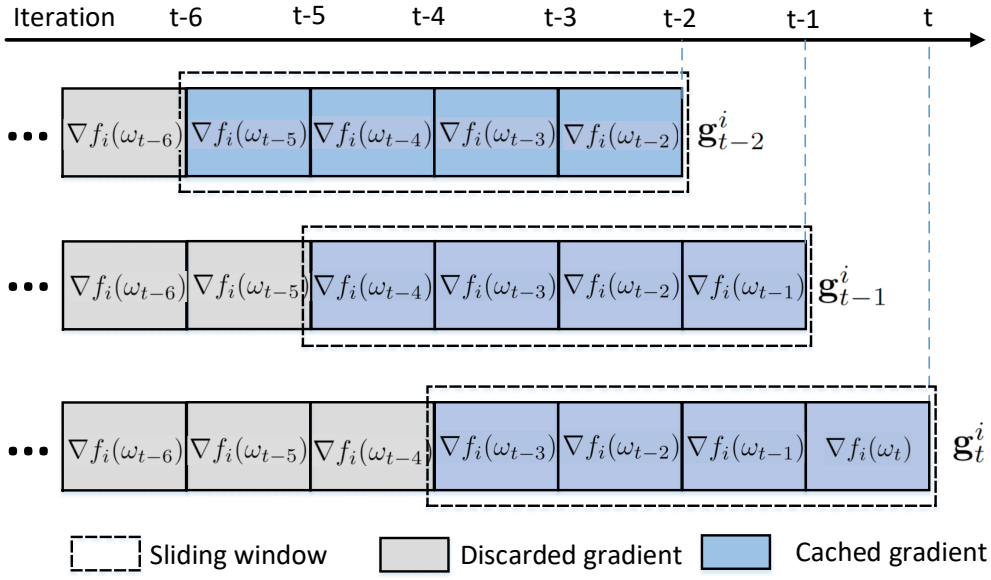


Figure 5. Illustration of an example with the sliding window.

The technique of sliding window solves the two challenges because: (1) Under a given scope, computing the average of multiple stochastic gradients in the sliding window is similar to increasing the batch size and thus results in the variance reduction; (2) The model training process of SGD is usually in an oscillated manner causing the sign of the many dimensions of the gradient drastically flip across the iterations, and as a consequence, the model difference generated by two adjacent iterations is sometimes even larger than that of several iterations. Computing the average of gradients from different models helps reduce such oscillation [10] and thus reduces the difference of gradient.

It is worthwhile to note that our method can also be combined with the other methods for the variance reduction, e.g., SVRG [35], SCSG [36], and SNVRG [37], to further improve the compression efficiency.

4.4. Implementation

The implementation workflow and architecture of DGSB are as follows. In a worker, the computing process consists of three layers: computing layer, optimizer, and sparsification layer. The computing layer is responsible for inputting the training data and computing the gradients (1) . For generality, we directly use the computing library of the existing learning systems, e.g., PyTorch, as this layer. The middle layer optimizer, consisting of various optimization algorithms, e.g., the general SGD, connects the sparsification layer, computing layer and the aggregation module in Parameter Server through three types of gradient: stochastic gradient, global gradient and sparsified differential gradient. With the stochastic gradient offered by optimizer, the sparsification layer first applies the function *difred* (2) to reduce the difference of the stochastic gradient and then computes the differential gradient (3). By using the threshold computed from the cached global gradient (4) in optimizer, the sparsification layer further sparsifies the differential gradient (5) and sends the sparsified differential gradient to the server (6). The main module in server is the aggregation module that recovers the integral gradients (7) and detects the correct gradient by using the Byzantine-resilient methods (8). After that, the server also computes the differential gradient

(9) and sends the non-zero dimensions to the workers (10). Finally, the optimizer layer in worker recovers the global gradient (11) and updates the model (12) to start the next iteration. The prototype is implemented on PyTorch.

5. Evaluations

In this section, we evaluate DGSB by training Logistic Regression (LR) model [38] and ResNet18 model [39], where they utilize MNIST dataset [40] and CIFAR-10 dataset [41], respectively. In all experiments, we set the learning rate to be 0.01, and the total number of workers (i.e. honest nodes and adversaries) to be 15. Additionally, we utilize vanilla non-compression SGD as the baseline. These controlled settings allow DGSB and all compared methods to be evaluated with identical models, datasets, aggregation operators, attack configurations, and training parameters. Our experiments are conducted on the public platform of Alibaba Cloud https://www.alibabacloud.com/ (accessed on 18 June 2026). Unless otherwise specified, the compression ratio denotes the percentage of transmitted non-zero gradient dimensions relative to the full gradient dimension; a lower compression ratio therefore indicates lower communication cost. When DGSB reports a single compression ratio, it is averaged over the training iterations.

We summarize the experimental conclusions based on the results of the empirical studies. First, we adjust the nominal implementation scale s to change the degree of compression in DGSB, and compare it with another two well-known sparsification methods (i.e. Mem-SGD [6] and GSpar [26]) under the same average compression ratio. The results show that a larger s leads to a smaller compression ratio and stronger communication compression, although it may slow down convergence. In terms of performance, DGSB is competitive with or better than the two gradient compression methods. Second, DGSB is compatible with various Byzantine-resilient methods (i.e. element-wise median [11] and FABA [9]), while Mem-SGD and GSpar are not. This incompatibility is reflected not only by non-convergence but also by much slower convergence under Byzantine attacks. Last but not least, as the sliding window stores more recent gradients, the communication cost decreases while the test accuracy remains stable.

5.1. Results of Non-Byzantine Case

We assume all workers are honest in the network and evaluate strongly-convex and non-convex models. Unlike Mem-SGD and GSpar, the compression ratio of DGSB varies from iteration to iteration. We set the nominal implementation scale s to 1, 5, and 10, which respectively refer to low, medium, and high communication compression. As explained in Section 4, the effective theoretical factor is $\lambda_t = s_t/N$, where the runtime scale satisfies $s_t \leq s$. Since all experiments use $N = 15$ workers, the corresponding effective factors are bounded above by $1/15$, $1/3$, and $2/3$, respectively, and thus satisfy the condition $\lambda_t < 1$ in Lemma 2. We then calculate the average compression ratio throughout the whole process and adopt it as the sparsification ratio for Mem-SGD and GSpar. The nominal scale s controls the communication–convergence trade-off rather than directly fixing the threshold, since the actual threshold H_t is updated in every iteration according to the norm of the cached global gradient. In practice, s can be selected as the smallest tested value that satisfies the target communication budget, thereby avoiding unnecessarily aggressive sparsification.

5.1.1. Communication Cost

The compression ratio of DGSB slightly increases at the beginning and eventually fluctuates around a stable level. A larger s results in a smaller compression ratio and therefore lower communication cost.

5.1.2. Convergence Rate

There is no distinct difference among three mentioned sparsification methods in terms of performance under the strongly-convex case. Besides, DGSB has the same convergence rate as the baseline (i.e. Vanilla SGD), which indicates that both differential operation and compression do not harm the convergence result under strongly-convex case.

As for a neural network, DGSB is slower to reach convergence under low and medium compression. However, from the perspective of convergence speed and convergence rate, DGSB outperforms GSpar in high compression.

5.2. Results of Byzantine Cases

A distributed system is vulnerable to Byzantine attacks. In this phase, we combine the evaluated compression methods with two Byzantine-resilient aggregation operators, element-wise median and FABA, to examine their compatibility. Here, B denotes the number of Byzantine workers. For each method, $\boldsymbol{v}$ denotes the message that a worker would normally upload: the full gradient for the non-compression baseline, a sparsified gradient for GSpar and Mem-SGD, and a sparsified differential gradient for DGSB. In the original mean-replacement attack, a Byzantine worker forges $\boldsymbol{v}'$ by replacing every non-zero element in $\boldsymbol{v}$ with the mean of its non-zero elements:

$$\boldsymbol{v} = [a_1, ..., a_n], \qquad \boldsymbol{v}' = [a'_1, ..., a'_n]$$

$$\bar{a} = \frac{a_1 + ... + a_n}{\text{len(non-zero elements in } \boldsymbol{v})}, \qquad a'_i = \begin{cases} \bar{a}, & a_i \neq 0 \\ 0, & a_i = 0 \end{cases}$$

We additionally consider sign-flipping and Gaussian-noise attacks. Let $\mathcal{S}(\boldsymbol{v}) = \{d \mid v_d \neq 0\}$ be the support of the upload vector. The sign-flipping attack reverses the uploaded vector without changing its magnitude:

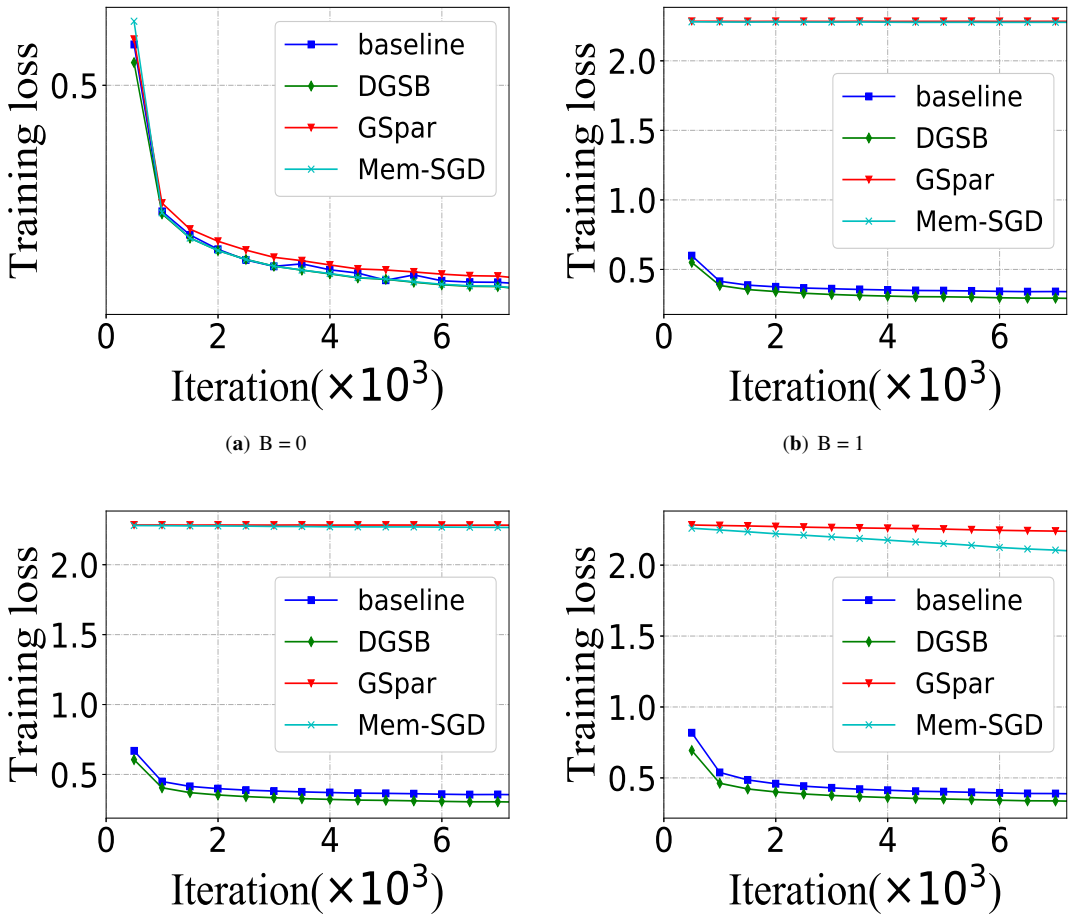
$$\boldsymbol{v}' = -\boldsymbol{v}.$$

For the Gaussian-noise attack, independently sampled noise is added only to the transmitted dimensions:

$$v'_d = \begin{cases} v_d + \xi_d, & d \in \mathcal{S}(\boldsymbol{v}), \\ 0, & d \notin \mathcal{S}(\boldsymbol{v}), \end{cases} \qquad \xi_d \sim \mathcal{N}(0, \sigma_{\boldsymbol{v}}^2), \qquad \sigma_{\boldsymbol{v}} = \frac{\|\boldsymbol{v}_{\mathcal{S}}\|_2}{\sqrt{|\mathcal{S}(\boldsymbol{v})|}}.$$

The noise is sampled independently for each Byzantine worker and iteration. Both additional attacks preserve the support of the original upload vector and therefore do not change its number of transmitted dimensions. We evaluate them using ResNet18 on CIFAR-10 with FABA, 15 workers, $B = 3$, and a learning rate of 0.01; all other settings follow the corresponding mean-replacement experiment.

Figure 6 presents the results when all compression methods train LR and use element-wise median to resist Byzantine attacks. When one or more Byzantine workers exist, GSpar and Mem-SGD remain at a high training loss, whereas DGSB outperforms the baseline. This result indicates that DGSB is compatible with element-wise median in strongly-convex cases, while GSpar and Mem-SGD are not.

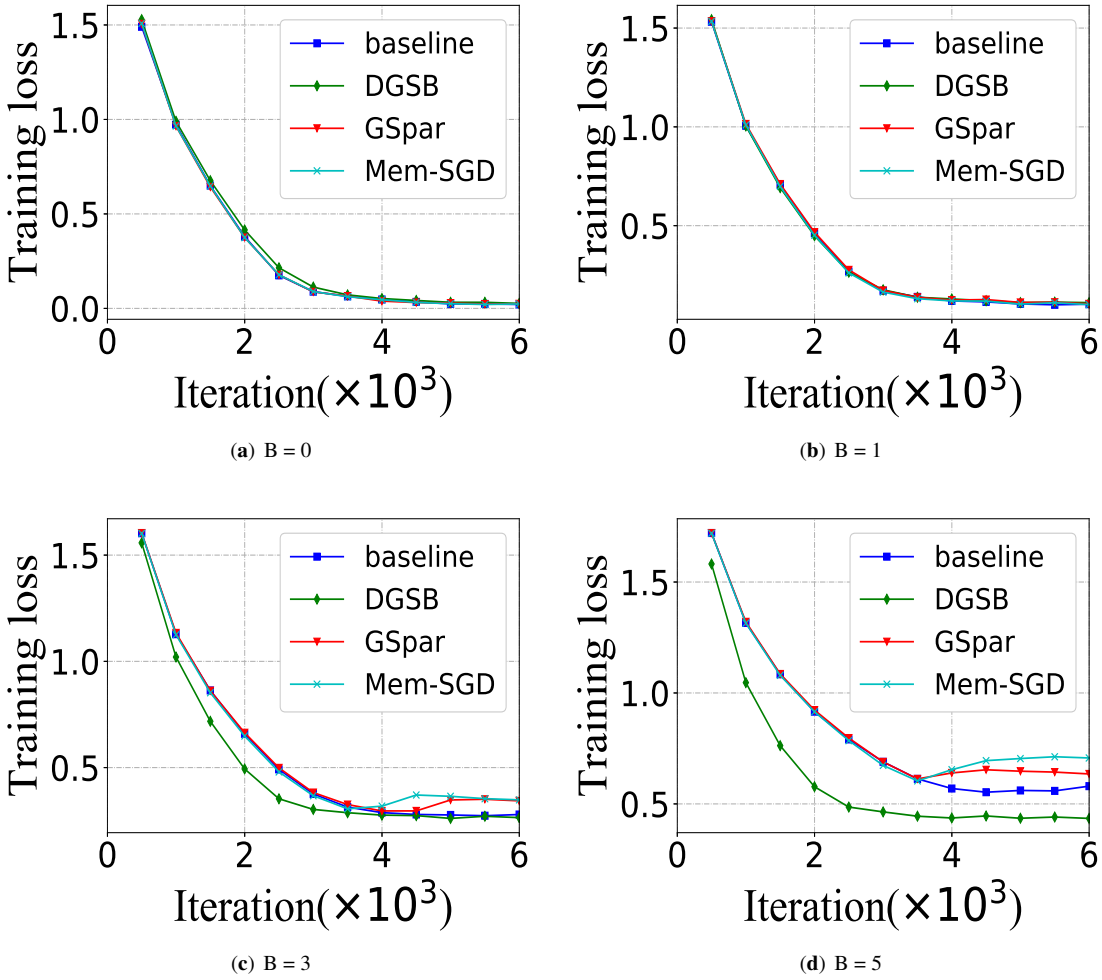


(**a**) B = 0 (**b**) B = 1

(**c**) B = 3 (**d**) B = 5

Figure 6. Training-loss results of logistic regression on MNIST with (**a**) $B = 0$; (**b**) $B = 1$; (**c**) $B = 3$; and (**d**) $B = 5$.

Figure 7 and Table 1 respectively show the training loss and test accuracy when the compression methods are combined with FABA for ResNet18. The methods perform similarly when only one Byzantine worker is present, whereas their differences become clearer as B increases. DGSB achieves the highest test accuracy in every evaluated Byzantine setting from $B = 1$ to $B = 5$. Averaged over these settings, DGSB reaches 72.19% accuracy, compared with 70.87% for the non-compression baseline, 71.11% for GSpar, and 70.67% for Mem-SGD.



(**a**) B = 0 (**b**) B = 1

(**c**) B = 3 (**d**) B = 5

Figure 7. Training-loss results of ResNet18 on CIFAR-10 with (**a**) $B = 0$; (**b**) $B = 1$; (**c**) $B = 3$; and (**d**) $B = 5$.

Table 2 reports the results under the three evaluated attacks at $B = 3$. DGSB achieves accuracies of 72.03%, 72.00%, and 71.94% under mean replacement, sign flipping, and Gaussian noise, respectively. The corresponding best competing results, all obtained by Mem-SGD, are 71.04%, 70.98%, and 70.90%. Thus, DGSB improves accuracy by 0.99–1.04 percentage points in these settings, and its accuracy varies by only 0.09 percentage points across the three attacks. These results show that the observed advantage is not limited to the mean-replacement construction, while making no claim about Byzantine attacks beyond those evaluated here.

Table 1. Test accuracy (%) of ResNet18 on CIFAR-10 with FABA: Mean-replacement attack with different numbers of Byzantine workers. The last column reports the average over $B = 1$–5.

Methods	**Number of Byzantine Workers**						**Avg.**
	$B = 0$	$B = 1$	$B = 2$	$B = 3$	$B = 4$	$B = 5$	$B = 1$–5
baseline	73.52	72.79	71.54	70.49	70.03	69.51	70.87
GSpar	72.82	73.01	71.50	70.93	70.69	69.42	71.11
Mem-SGD	**73.83**	72.67	71.27	71.04	70.24	68.13	70.67
DGSB	73.54	**73.22**	**72.17**	**72.03**	**71.96**	**71.55**	**72.19**

Note: Bold values indicate the best result in each column.

Table 2. Test accuracy (%) of ResNet18 on CIFAR-10 with FABA: Different attack types with fixed $B = 3$.

Methods	Mean Replacement	Sign Flipping	Gaussian Noise
baseline	70.49	70.56	70.42
GSpar	70.93	70.87	70.78
Mem-SGD	71.04	70.98	70.90
DGSB	**72.03**	**72.00**	**71.94**

Note: Bold values indicate the best result in each column.

5.3. Impact of Sliding Window

Figure 8 demonstrates the impact of the sliding-window size E on DGSB under both non-Byzantine and Byzantine cases with ResNet18. The average compression ratio decreases consistently as E increases, while the final test accuracy remains within 70%–75% and exhibits no monotonic degradation. Thus, E provides a direct communication–memory trade-off rather than requiring an accuracy-specific optimum for every setting. A moderate value can be selected according to the available worker memory; for example, $E = 5$ achieves compression ratios of approximately 4.13% and 4.30% with test accuracies of 74.32% and 72.44% in the non-Byzantine and Byzantine cases, respectively.

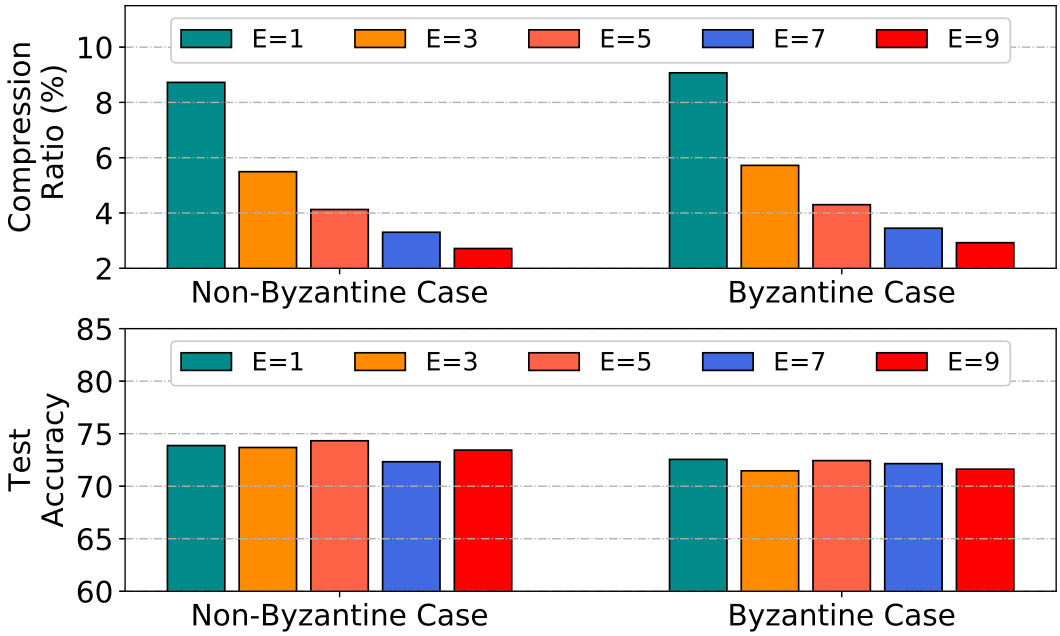


Figure 8. Investigation of the impact of the sliding-window size. The upper figure shows the average compression ratio under different window sizes. The bottom figure shows the final test accuracy under different window sizes.

6. Conclusions

In this paper, we propose a new communication compression method named DGSB for distributed learning in the presence of Byzantine workers. The core idea of DGSB is to sparsify the differential gradient in the worker and recover the original gradient in the server, such that it can be applied with existing Byzantine-resilient methods. Through extensive evaluations under multiple settings, DGSB achieves significant communication reduction while maintaining robust training performance when Byzantine workers exist. The convergence analysis isolates the compression and reconstruction mechanism under non-Byzantine Mean aggregation, while compatibility with robust aggregation is evaluated empirically under the stated threat model. Extending the convergence analysis to specific Byzantine-resilient aggregation operators is an important direction for future work. The current design and evaluation assume synchronous iterations. Extending DGSB to asynchronous execution would require version-aware reconstruction for stale or out-of-order differential updates and a separate convergence analysis. Evaluations with larger models, additional worker scales, and non-IID data are also left for future work.

Author Contributions

Z.J.: Conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing—original draft preparation, visualization; H.W.: Conceptualization, methodology, writing—review and editing, supervision, project administration, funding acquisition; S.L.: validation, writing—review and editing; Z.Z.: validation, writing—review and editing; R.L.: supervision, funding acquisition, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding

This research was funded by the National Key Research and Development Program of China under grant 2024YFC3307900; the National Natural Science Foundation of China under grants 62302184, 62376103, 62436003 and 62206102; Major Science and Technology Project of Hubei Province under grant 2024BAA008; Hubei Science and Technology Talent Service Project under grant 2024DJC078; Ant Group through CCF-Ant Research Fund; and Fundamental Research Funds for the Central Universities under grant YCJJ20252319. Computations were performed on the HPC Platform of Huazhong University of Science and Technology.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

The MNIST dataset analyzed in this study is publicly available from the UCI Machine Learning Repository, and the CIFAR-10 dataset is publicly available from the official CIFAR-10 dataset page. No new datasets were generated during this study.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

During the proofreading and typesetting review of this manuscript, the authors used an AI-assisted proofreading tool solely to check formatting consistency and typographical issues. No AI tool was used to write the manuscript or generate its scientific content, analysis, results, or conclusions. The authors reviewed all suggested changes and take full responsibility for the content of the published article.

Appendix A. Proofs of Lemmas

Appendix A.1. Proof of Lemma 1

Proof. For worker i, define the compression error

$$\mathbf{e}_t^i = \mathbf{g}_t^i - \tilde{\mathbf{g}}_t^i = \left(\mathbf{g}_t^i - \tilde{\mathbf{g}}_{t-1}^i\right) - \left(\tilde{\mathbf{g}}_t^i - \tilde{\mathbf{g}}_{t-1}^i\right) = \Delta_t^i - \tilde{\Delta}_t^i.$$

In the non-Byzantine case with Mean aggregation, the update is

$$\omega_{t+1} = \omega_t - \eta\tilde{\mathbf{g}}_t, \qquad \tilde{\mathbf{g}}_t = \frac{1}{P}\sum_{i=1}^{P}\tilde{\mathbf{g}}_t^i = \bar{\mathbf{g}}_t - \bar{\mathbf{e}}_t,$$

where $\bar{\mathbf{g}}_t = \frac{1}{P}\sum_{i=1}^{P}\mathbf{g}_t^i$ and $\bar{\mathbf{e}}_t = \frac{1}{P}\sum_{i=1}^{P}\mathbf{e}_t^i$.

By L-smoothness of F,

$$F(\omega_{t+1}) - F(\omega_t) \leq -\eta\langle\nabla F(\omega_t), \bar{\mathbf{g}}_t - \bar{\mathbf{e}}_t\rangle + \frac{L\eta^2}{2}\|\bar{\mathbf{g}}_t - \bar{\mathbf{e}}_t\|_2^2.$$

Taking expectation and using unbiasedness $\mathbb{E}[\bar{\mathbf{g}}_t] = \nabla F(\omega_t)$, we get

$$\begin{aligned}
&\mathbb{E}[F(\omega_{t+1}) - F(\omega_t)] \\
&\leq -\eta \|\nabla F(\omega_t)\|_2^2 + \eta\, \mathbb{E}\langle \nabla F(\omega_t), \bar{\mathbf{e}}_t \rangle + \frac{L\eta^2}{2} \mathbb{E}\|\bar{\mathbf{g}}_t - \bar{\mathbf{e}}_t\|_2^2 \\
&\leq -\eta \|\nabla F(\omega_t)\|_2^2 + \eta\, \mathbb{E}\langle \nabla F(\omega_t), \bar{\mathbf{e}}_t \rangle + \frac{L\eta^2}{2} \mathbb{E}\|\bar{\mathbf{g}}_t\|_2^2 - L\eta^2 \mathbb{E}\langle \nabla F(\omega_t), \bar{\mathbf{e}}_t \rangle + \frac{L\eta^2}{2} \mathbb{E}\|\bar{\mathbf{e}}_t\|_2^2.
\end{aligned}$$

Since workers use independent mini-batches, by Assumption 1 (unbiasedness and bounded variance),

$$\mathbb{E}\|\bar{\mathbf{g}}_t\|_2^2 = \|\nabla F(\omega_t)\|_2^2 + \mathbb{E}\|\bar{\mathbf{g}}_t - \nabla F(\omega_t)\|_2^2 \leq \|\nabla F(\omega_t)\|_2^2 + \frac{\sigma^2}{P}.$$

Therefore,

$$\begin{aligned}
&\mathbb{E}[F(\omega_{t+1}) - F(\omega_t)] \\
&\leq \eta \left(\frac{L\eta}{2} - 1 \right) \|\nabla F(\omega_t)\|_2^2 + \eta(1 - L\eta) \mathbb{E}\langle \nabla F(\omega_t), \bar{\mathbf{e}}_t \rangle + \frac{L\eta^2}{2} \mathbb{E}\|\bar{\mathbf{e}}_t\|_2^2 + \frac{L\eta^2\sigma^2}{2P}.
\end{aligned}$$

Because $\eta \leq 1/L$, we have $1 - L\eta \geq 0$, and by Young's inequality $\langle a, b \rangle \leq \frac{1}{2}(\|a\|_2^2 + \|b\|_2^2)$,

$$\mathbb{E}[F(\omega_{t+1}) - F(\omega_t)] \leq \eta \left[-\frac{1}{2} \|\nabla F(\omega_t)\|_2^2 + \frac{1}{2} \mathbb{E}\|\bar{\mathbf{e}}_t\|_2^2 \right] + \frac{L\eta^2\sigma^2}{2P}.$$

Also, $\|\bar{\mathbf{e}}_t\|_2^2 \leq \frac{1}{P} \sum_{i=1}^{P} \|\mathbf{e}_t^i\|_2^2$. Hence,

$$\mathbb{E}[F(\omega_{t+1}) - F(\omega_t)] \leq \frac{\eta}{P} \sum_{i=1}^{P} \mathbb{E}\left[-\frac{1}{2} \|\nabla F(\omega_t)\|_2^2 + \frac{1}{2} \|\mathbf{e}_t^i\|_2^2 \right] + \frac{L\eta^2\sigma^2}{2P}.$$

Summing from $t = 1$ to T, and using $\mathbf{e}_t^i = \Delta_t^i - \tilde{\Delta}_t^i$, gives (10). □

Appendix A.2. Proof of Lemma 2

Proof. Let $\mathbf{e}_t^i = \Delta_t^i - \tilde{\Delta}_t^i$. From (11), for each t, i,

$$\|\mathbf{e}_t^i\|_2^2 = \sum_{d=1}^{D} |e_t^{i,d}|^2 \leq \sum_{d=1}^{D} \frac{\lambda_t^2}{D} \|\nabla F(\omega_t)\|_2^2 = \lambda_t^2 \|\nabla F(\omega_t)\|_2^2.$$

Substituting this bound into Lemma 1, we get

$$\mathbb{E}F(\omega_{T+1}) - F(\omega_1) \leq \frac{\eta}{P} \sum_{t=1}^{T} \sum_{i=1}^{P} \mathbb{E}\left[\frac{\lambda_t^2 - 1}{2} \|\nabla F(\omega_t)\|_2^2 \right] + \frac{L\eta^2\sigma^2 T}{2P}.$$

Let $\lambda = \sup_t \lambda_t < 1$, then

$$\mathbb{E}F(\omega_{T+1}) - F(\omega_1) \leq \eta \sum_{t=1}^{T} \frac{\lambda^2 - 1}{2} \mathbb{E}\|\nabla F(\omega_t)\|_2^2 + \frac{L\eta^2\sigma^2 T}{2P}.$$

Since $F(\omega) \geq F_*$, rearranging yields

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\|\nabla F(\omega_t)\|_2^2 \leq \frac{2\big(F(\omega_1) - F_*\big)}{\eta T(1 - \lambda^2)} + \frac{L\eta\sigma^2}{P(1 - \lambda^2)}.$$

Thus Algorithm 1 has bounded stationarity error and converges (to a neighborhood for constant η, and to a stationary point under diminishing η). □

References

1. Harlap, A.; Tumanov, A.; Chung, A.; et al. Proteus: Agile ML Elasticity Through Tiered Reliability in Dynamic Resource Markets. In Proceedings of the Twelfth European Conference on Computer Systems (EuroSys), Belgrade, Serbia, 23–26 April 2017; pp. 589–604.
2. Konečný, J.; McMahan, H.B.; Yu, F.X.; et al. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv* **2016**, arXiv:1610.05492.
3. Li, M.; Andersen, D.G.; Park, J.W.; et al. Scaling Distributed Machine Learning with the Parameter Server. In Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, Broomfield, CO, USA, 6–8 October 2014; pp. 583–598.
4. Aji, A.F.; Heafield, K. Sparse Communication for Distributed Gradient Descent. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), Copenhagen, Denmark, 7–11 September 2017; pp. 440–445.
5. Alistarh, D.; Hoefler, T.; Johansson, M.; et al. The Convergence of Sparsified Gradient Methods. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 5977–5987.
6. Stich, S.U.; Cordonnier, J.; Jaggi, M. Sparsified SGD with Memory. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 4452–4463.
7. Blanchard, P.; Mhamdi, E.M.E.; Guerraoui, R.; et al. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 118–128.
8. Chen, Y.; Su, L.; Xu, J. Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Irvine, CA, USA, 18–22 June 2018; p. 96.
9. Xia, Q.; Tao, Z.; Hao, Z.; et al. FABA: An Algorithm for Fast Aggregation Against Byzantine Attacks in Distributed Neural Networks. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI), Macao, China, 10–16 August 2019; pp. 4824–4830.
10. Ruder, S. An Overview of Gradient Descent Optimization Algorithms. *arXiv* **2016**, arXiv:1609.04747.
11. Yin, D.; Chen, Y.; Ramchandran, K.; et al. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 5650–5659.
12. Xie, C.; Koyejo, O.; Gupta, I. Generalized Byzantine-Tolerant SGD. *arXiv* **2018**, arXiv:1802.10116.
13. Alistarh, D.; Allen-Zhu, Z.; Li, J. Byzantine Stochastic Gradient Descent. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 4618–4628.
14. Chen, L.; Wang, H.; Charles, Z.; et al. DRACO: Byzantine-Resilient Distributed Training via Redundant Gradients. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 903–912.
15. Li, L.; Xu, W.; Chen, T.; et al. RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 1544–1551.
16. Damaskinos, G.; Mhamdi, E.M.E.; Guerraoui, R.; et al. Asynchronous Byzantine Machine Learning (the Case of SGD). In Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 1145–1154.
17. Yang, Y.; Li, W. BASGD: Buffered Asynchronous SGD for Byzantine Learning. *arXiv* **2020**, arXiv:2003.00937.
18. Liu, Y.; Chen, C.; Lyu, L.; et al. Byzantine-Robust Learning on Heterogeneous Data via Gradient Splitting. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 21404–21425.
19. Allouah, Y.; Guerraoui, R.; Gupta, N.; et al. Adaptive Gradient Clipping for Robust Federated Learning. In Proceedings of the 13th International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
20. Hsieh, K.; Harlap, A.; Vijaykumar, N.; et al. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, USA, 27–29 March 2017; pp. 629–647.
21. Chen, C.; Choi, J.; Brand, D.; et al. AdaComp: Adaptive Residual Gradient Compression for Data-Parallel Distributed Training. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 2827–2835.
22. Shi, S.; Zhao, K.; Wang, Q.; et al. A Convergence Analysis of Distributed SGD with Communication-Efficient Gradient Sparsification. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI), Macao, China, 10–16 August 2019; pp. 3411–3417.

23. Sattler, F.; Wiedemann, S.; Müller, K.; et al. Robust and Communication-Efficient Federated Learning from Non-i.i.d. Data. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 3400–3413.
24. Lin, Y.; Han, S.; Mao, H.; et al. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In Proceedings of the 6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
25. Zhao, S.; Gao, H.; Li, W. On the Convergence of Memory-Based Distributed SGD. *arXiv* **2019**, arXiv:1905.12960.
26. Wangni, J.; Wang, J.; Liu, J.; et al. Gradient Sparsification for Communication-Efficient Distributed Optimization. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS), Montréal, QC, Canada, 3–8 December 2018; pp. 1299–1309.
27. Li, S.; Xu, W.; Wang, H.; et al. FedBAT: Communication-Efficient Federated Learning via Learnable Binarization. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024; Volume 235, pp. 29074–29095.
28. Kim, D.Y.; Han, D.J.; Seo, J.; et al. Achieving Lossless Gradient Sparsification via Mapping to Alternative Space in Federated Learning. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024; Volume 235, pp. 23867–23900.
29. Li, S.; Luo, X.; Wang, H.; et al. The Panaceas for Improving Low-Rank Decomposition in Communication-Efficient Federated Learning. In Proceedings of the 42nd International Conference on Machine Learning (ICML), Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 35536–35561.
30. Wang, H.; Liu, X.; Niu, J.; et al. Why Go Full? Elevating Federated Learning Through Partial Network Updates. In Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 9–15 December 2024; pp. 99773–99799.
31. Rammal, A.; Gruntkowska, K.; Fedin, N.; et al. Communication Compression for Byzantine Robust Learning: New Efficient Algorithms and Improved Rates. In Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS), Valencia, Spain, 2–4 May 2024; Volume 238, pp. 1207–1215.
32. Zhou, F.; Cong, G. On the Convergence Properties of a K-Step Averaging Stochastic Gradient Descent Algorithm for Nonconvex Optimization. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden, 13–19 July 2018; pp. 3219–3227.
33. Zhang, X.; Wang, J.; Joshi, G.; et al. Machine Learning on Volatile Instances. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Online, 6–9 July 2020; pp. 139–148.
34. PyTorch. PyTorch MNIST Example. Available online: https://github.com/pytorch/examples/tree/main/mnist (accessed on 15 June 2026).
35. Johnson, R.; Zhang, T. Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NeurIPS), Lake Tahoe, NV, USA, 5–10 December 2013; pp. 315–323.
36. Lei, L.; Jordan, M.I. Less Than a Single Pass: Stochastically Controlled Stochastic Gradient. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 20–22 April 2017; Volume 54, pp. 148–156.
37. Zhou, D.; Xu, P.; Gu, Q. Stochastic Nested Variance Reduction for Nonconvex Optimization. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS), Montréal, QC, Canada, 3–8 December 2018.
38. Hosmer, D.W.; Hosmer, T.; Le Cessie, S.; et al. A Comparison of Goodness-of-Fit Tests for the Logistic Regression Model. *Stat. Med.* **1997**, *16*, 965–980.
39. He, K.; Zhang, X.; Ren, S.; et al. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
40. LeCun, Y.; Cortes, C.; Burges, C.J.C. MNIST Database of Handwritten Digits [Dataset]. UCI Machine Learning Repository, 1998. Available online: https://archive.ics.uci.edu/dataset/683/mnist (accessed on 28 June 2026).
41. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; University of Toronto: Toronto, ON, Canada, 2009. Available online: https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 28 June 2026).