Design of a Quantization-based DNN Delta Compression Framework for Model Snapshots and Federated Learning

Haoyu Jin, Donglei Wu, Shuyu Zhang, Xiangyu Zou, Sian Jin, Dingwen Tao, Qing Liao and Wen Xia

Abstract—Deep neural networks (DNNs) have achieved remarkable success in many fields. However, large-scale DNNs also bring storage costs when storing snapshots for preventing clusters' frequent failures or incur significant communication overheads when transmitting DNNs in the Federated Learning (FL). Recently, several approaches, such as Delta-DNN and LC-Checkpoint, aim to reduce the size of DNNs' snapshot storage by compressing the difference between two neighboring versions of the DNNs (a.k.a., delta). However, we observe that existing approaches, applying traditional global lossy quantization techniques in DNN's delta compression, can not fully exploit the data similarity since the parameters' value ranges vary among layers. To fully explore the similarity of the delta model and improve the compression ratio, we propose a quantization-based local-sensitive delta compression approach, named QD-Compressor, by developing a layer-based local-sensitive quantization scheme and error feedback mechanism. Specifically, the quantizers and number of quantization bits are adaptive among layers based on the value distribution and weighted entropy of the delta's parameters. To avoid quantization error degrading the performance of the restored model, an alternative error feedback mechanism is designed to dynamically correct the quantization error during the training process. Experiments on multiple popular DNNs and datasets show that QD-Compressor obtains a higher 7×-40× compression ratio in the model snapshot compression scenario than the state-of-the-art approaches. Additionally, QD-Compressor achieves an 11×-15× compression ratio to the residual model of the Federated Learning compression scenario.

Index Terms—neural networks, quantization, delta compression, snapshot, distribution learning.

1 Introduction

Over the past decades, Deep Neural Networks (DNNs) have achieved significant improvements in a wide spectrum of application domains, such as image classification [1], [2], [3], object detection, recognition [4], [5], semantic segmentation [6], face tracking and alignment [7], [8], etc. As these tasks become more and more challenging, the network becomes more and more complex by increasing the depth and width, which results in a large number of parameters and high computational complexity for better performance. For example, AlexNet [1] has 61 million parameters that need 249 MB of memory and costs 1.5 billion operations to classify one image, VGG-19 [9] even has 144 million parameters.

With the development of artificial intelligence research, the intermediate models (e.g., the snapshots and the residual model) of the under-training neural networks are required to store the local. For example, (1) more and more studies employ **model snapshots** for scientific analysis [10], [11] and ensemble learning [12], [13] to advance the model training performance. (2) In the resource-constrained Federated Learning clients (e.g., IoT devices, mobile phones, and wearable devices), the Top-K sparsification-based Federated Learning compression is efficient in reducing communication costs [14], [15]. Generally, the dropped parameters generated by the Top-K sparsification (i.e., **residual model**) are required to be stashed locally to keep the model accuracy. With the size of DNNs is increasingly larger, however, the storage challenge of these intermediate results becomes more severe as there is the same size of the full neural networks.

To facilitate the applications of DNNs, neural network model compression have become a popular topic in the research and industry communities. There are already several techniques aim at reducing the size of a single neural network, including pruning [16], knowledge distillation [17], and quantization [18], [19], [20]. Recently, the similarity in the neighboring epochs of the trained network has been utilized to advance the Delta compression performance [21], [22]. These methods first *convert the global similarity to compressibility* by imposing the global differences quantization techniques on the floating delta model and then achieve a considerable compression ratio on this quantized delta. **However, we observe three limitations of these global differences in quantization-based methods in this paper:** (1) The global mapping of the floating-point parameters to

- *H. Jin, D. Wu, S. Zhang, X. Zou are with Harbin Institute of Technology, Shenzhen, Guangdong 518055, China. E-mail: jinhy549@gmail.com, {donglei.wu, xiangyu.zou, shuyu.zhang97}@hotmail.com.*
- *S. Jin, D. Tao is with Indiana University Bloomington, Pullman, WA, 47405. E-mail: {sianjin, ditao}@iu.edu.*
- *Q. Liao and W. Xia are with the Harbin Institute of Technology, Shenzhen, Peng Cheng Laboratory, Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen, Guangdong 518055, China. E-mail: {liaoqing, xiawen}@hit.edu.cn.*

*The preliminary manuscript was published in IEEE ICCD'21. In the journal version, we further improve the compression ratio (about 40% higher on average) by developing a **Weighted Entropy**-based strategy that assigns different layers to various number of quantization bits. We also verify the effectiveness of our method in the two realistic applications: snapshot recovery from the training crash and residual model compression in federated learning. Moreover, we get additional measurement results from our analysis and experiments.*

integers version with the same quantizer (i.e., the global quantization) can not fully exploit the data similarity between them for delta compression. (2) Quantizing the entire network with the same number of bits (a.k.a., bit width) ignores the differences of parameter distributions among layers. (3) Additionally, the post-processing operations (e.g., error-bound selection and re-training) are required to maintain the accuracy of the decompressed model, which means additional computation and time cost.

To further prove the existence of the above problems, we verify multiple popular DNNs and observe that (1) parameters in a layer usually are near zero, but the value range of parameters is not always similar. Therefore, the naive quantization with a global quantizer to the Delta model will limit the overall compression ratio since the similarity cannot be fully exploited. (2) The quantization errors produced in the lossy delta compression will degrade the test accuracy of the restored model. To restore the target accuracy for the restored model, additional pre or post-processing is required, which incurs additional overhead.

Motivated by the above observations, we propose a novel quantization-based delta compression framework **QD-Compressor** involves two core schemes, *Local-Sensitive Quantization* and *Error Feedback Mechanism*, which significantly improves the communication ratio without degrading the model quality. Broadly speaking, instead of first calculating the delta data (i.e., the difference) on the floating-point parameters and then globally quantizing the floating-point delta data of DNNs, QD-Compressor's local-sensitive quantization scheme significantly improves compressibility of the delta data by first quantizes DNNs and then calculates delta data upon the quantized integer networks. Additionally, local-sensitive quantization sets the (1) quantizers and (2) bit width adaptive to each layer according to the parameter's value ranges and weighted entropy (not globally). In this way, the local-sensitive quantization scheme achieves a higher compression ratio by the layer-based adaptive compression mechanism, which will be detailed in Section 3. Further, the error feedback mechanism can correct the quantization errors in the training process, which helps eliminate the accuracy loss of the restored model.

In summary, QD-Compressor significantly improves the compression ratio for the delta model by employing a **weighted entropy-based local-sensitive quantization scheme**. Meanwhile, with the support of the **error feedback mechanism**, the model accuracy of the restored model can be well maintained in QD-compressor.

Generally, contributions of this paper are four-folds:

- We **observe** that: (1) The value ranges and the weighted entropy of parameters vary among layers, and parameters in the same layers are very close. (2) The traditional global quantization schemes (e.g., Delta-DNN) quantize each layer using a unified quantizer and bit number for each layer but cannot fully exploit the similarity between neighboring versions of a neural network. (3) Existing quantization approaches may lead to accuracy loss on DNNs, thereby incurring additional costs to fine-tune the restored model.
- We **propose** a novel delta compression framework called QD-Compressor. It develops a novel weighted entropy-based local-sensitive quantization to adaptively vary *Quantizers* and *Bit Number* for different layers in DNNs. It also calculate the high compressible delta after quantization to achieve a significant compression ratio. Further, an **Error Feedback** mechanism is designed by introducing the quantization error into the training process to reduce the accuracy loss of the restored model.
- We **implement** our proposed QD-Compressor in two realistic scenarios: (1) Top-k sparsification Federated Learning compression and (2) model snapshots recovery for the model training crash. Evaluations on six popular DNNs suggest that QD-Compressor achieves a 7×-40× higher compression ratio while well maintaining the model accuracy, compared with the state-of-the-art compressors (i.e., SZ, Zstd, LC-Checkpoint, and Delta-DNN).

The rest of the paper is organized as follows. Section 2 presents the background and related works of this paper. Section 3 introduces the observations and motivations. Section 4 describes the design methodologies of QD-Compressor framework in detail. Section 6 discusses the evaluation results of QD-Compressor on six well-known DNNs, compared with state-of-the-art compressors and shows the evaluation results of QD-Compressor in the scenarios of distribution learning compression and snapshot recovery. In Section 7, we conclude this paper.

2 Background and Related works

In this section, we present the necessary background of traditional data compression techniques and model compression research. Then, we introduce typical application scenarios for lossy delta compression in neural networks.

2.1 Data Compression Techniques

Data compression, aiming at reducing data scale, has been a traditional technology for decades. Generally, data compression techniques can be categorized into three classes: general compression, delta compression, and data deduplication.

General compression techniques [23], [24], [25] focus on file-level workload and compresses data at byte level by entropy coding [26], dictionary coding [27], and other techniques. When facing large-scale storage, general compression is not feasible because its speed is relatively slow, and it only can eliminate redundancies in limited windows [28]. To address this challenge, **data deduplication** [29] is proposed, which splits data into chunks, and deduplicates identical ones. Data deduplication achieves a much higher speed and can detect redundancies in a large system, but it is a coarse-grained approach and can not fully exploit compressibility within workloads. Therefore, **delta compression** [30], [31] is designed to bridge the gap between fine-grained general compression and course-grained data compression, which picks up not identical but similar chunks and eliminates redundancies between them. In summary, traditional compression, delta compression, and data deduplication are all lossless approaches.

Besides, in HPC (high-performance computing) fields, there is usually an amount of floating-point data produced,

which brings challenges to the storage system. For floating-point data produced by HPC, the storage way of floating-point numbers makes it difficult to find two storage blocks with exactly the same storage format. Floating-point data usually have a large information entropy and are hard to be compressed by lossless approaches. Thus, **lossy compression** [32], [33], [34], [35] techniques are proposed, which compresses floating-point data with a user-specific error bound. Recently, an algorithm for HPC data compression called SZ [33], [34] proposed to improve the compression ratio of floating-point numbers. SZ takes the logarithm of the data to convert the control range of the relative error into the control range of the absolute error, and then compresses the data by using a designed predictor to represent subsequent values. The compression ratio of SZ is not only dependent on the distribution characteristics of the data but also related to the performance of the predictor. There will be errors in decompressed data in these lossy compression approaches, but they also promise the errors will always be smaller than the predefined error bound.

2.2 Neural Network Compression Techniques

Nowadays, there are more and more complicated tasks that can be well solved by neural networks which caused the super development of the neural network. But, neural networks become deeper and wider with the more difficult task which needs not only high computational abilities but also large storage space. As a result, model compression techniques are necessary to make better use of those massive networks. Scaling up the size of Deep Neural Networks (DNNs) (e.g., width, depth, etc.) is known to effectively improve model accuracy. But large model size impedes training on resource-constrained devices, thus typical DNNs compression methods, *pruning* and *quantization* gain increased attention in recent years.

Pruning is to remove the 'unimportant' parameters of DNNs with specific rules, which effectively decreases the complexity of the network and avoids the overfitting issue. Pruning-based methods consist of two categories: *Non-structured* pruning and *structured* pruning. *non-structured* means the arbitrary weight in the network can be pruned [16]. *Structured* pruning considers the matrix format of parameters with indices and prune the whole filter or channel [36]. However, the problem they all face is how to determine the importance of parameters or filters to prune. Additionally, pruning will change the structure of the network. Liu *et al.* [37] point out that the structure of the network after pruning is more important than the weight, which means that pruning is helpful for network structure to search to design a new model.

Quantization technique is to map the parameters of the network from the floating-point numbers into lower bit-depth representations. Compared with the pruning method, the quantization technique assumes that it does not need so many bits to store the parameter of a network and design a new representation with low bit-depth to save the storage space of a parameter.

Considering the trade-off between compression ratio and accuracy, typical quantization methods are 8-bit quantization and binarized. Generally, 8-bit quantization converts 32-bit floating-point number to 8-bit integer. Meanwhile, the float-arithmetic calculation is replaced with the 8-bit integer-arithmetic calculation. Jacob *et al.* [18] use linear mapping to convert the floating-point number to 8-bit integer and training with simulated quantization to minus the accuracy loss of quantization. Compared to the 8-bit quantization, Binarized [19], [20], [38] can be regarded as a special extreme quantization method that only uses 1 bit to represent the weight or activation. The key point of binary quantization is to use bit operations to replace multiply and add operations.

Recently, Delta-DNN [21] notices the similarity between neighboring versions of a neural network, compresses two neural networks by calculating their differences, and achieves a 2×-10× higher compression ratio than the traditional model compression approaches. Specifically, Delta-DNN first calculates the differences (i.e., the delta data) of corresponding parameters and then applies a global quantization scheme on the delta data to expose more compressibility. Finally, the quantized delta data will be compressed by lossless compression techniques.

3 Observation and Motivation

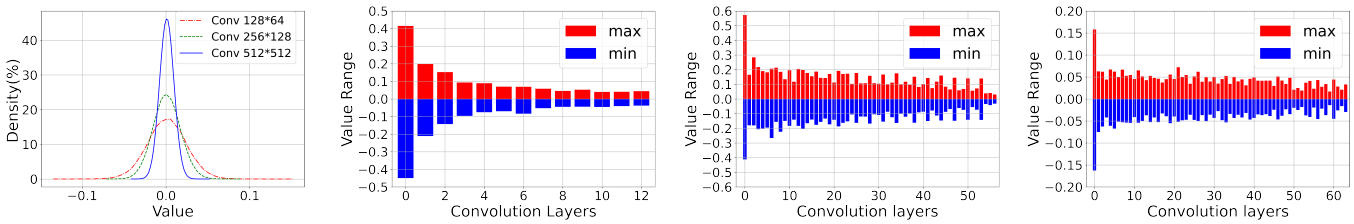
As introduced earlier, the **similarity** between versions of a neural network has been observed and utilized for saving resources in storage space and internet traffic [21]. Besides, since floating-point parameters are hard to be compressed, **quantization** is a widely used lossy compression technique [33], [34], [35], which converts the floating-point data with little compressibility to the integer data with higher compressibility. Recently, approaches [21], [22] have applied the global quantization technique to the floating-point differences of two neighboring epochs of neural networks (i.e., delta data), effectively improving the compression ratio than the traditional floating-point data compression techniques.

Problems of the quantization strategy in existing methods: Generally, determining the quantizer under a large value leads to a high compression ratio but brings serious quantization errors for small data, while fine-grained quantization achieves opposite results. Due to the value ranges of corresponding parameters usually being large and distributed randomly, the widely used global quantization (i.e., using the same quantizer and bit width for the entire model) cannot fully exploit the similarity for compression. It is hard to select one suitable global quantization granularity and quantization level (i.e., quantizer and bit width).

Besides, few existing approaches consider repairing the quantization errors during the training process, which leads to a degraded model accuracy in the restored model [21], [39]. Additional time and computational overhead are incurred to recover the target accuracy for the restored model.

To address these challenges, we study the parameter characteristics of DNNs and learn some observations:

1) **Figure 1** (a) shows that the parameter distribution is basically symmetrical about zero, and the variance of parameters is small. That is to say, the overall value range (i.e., magnitude) of the parameters in the neural network is relatively small. However, since the value range of whole parameters is relatively large (i.e., long-tailed), the use of floating-point values to express the value range of parameters is very large.



(a) Parameter distribution of convolution layers in VGG-16 (b) Parameter range of VGG-16 (c) Parameter range of MobileNet (d) Parameter range of GoogleNet

Fig. 1. Distribution and range of parameters among convolution layers in neural networks.

Using a representation with a big value range to represent a narrow value range will produce information redundancy.

2) As the examples shown in **Figure 1** (b) (c) (d), the value ranges of parameters in different layers vary greatly. The whole value range of DNN's parameters is very large compared with the small value range in some layers (near 20×). Directly calculating the difference (i.e., delta data) on the original floating-point version of neighboring networks (as Delta-DNN does) not only make it hard to determine an appropriate quantizer for all parameters but also limits the compressibility of the delta data.
3) The quantization errors produced in the compression of each version of the network will degrade the inference accuracy of the restored model. Existing works compensate for the quantization error by designing the pre- or post-processing operation (e.g., Error-bound Selection, fine-tuning) [21], [39]. However, additional operations for accuracy compensation always lead to more time and computation overhead.

According to above observations, we get two important motivations for this work:

Motivation 1: *Observation 1* and *Observation 2* allow us to explore a better quantization strategy for a high compression ratio. More specifically, the parameters' value range is very different among layers, while the value range of delta data is also very large. Globally quantizing the floating-point difference (i.e., delta data) will obtain a limited compression ratio since it needs to maintain the model inference accuracy meanwhile. To solve the above dilemma and achieve better compression performance, we are motivated to develop a more efficient layer-based **local-sensitive quantization scheme** by allocating different bit width and quantizer for each layer.

Motivation 2: *Observation 3* suggests that additional operations are always required to reduce the impact of model quantization errors and maintain model accuracy. Delta-DNN uses an additional post-processing operation called error-bound selection to keep decompressed model's inference accuracy, which necessitates additional calculation and time. LC-Checkpoint recovers the model accuracy by incurring fine-tuning process, which also means more time and computation overhead. To this end, we combine the compression process and training process by introducing an **error feedback scheme to dynamically correct the accuracy loss in each training round**. To do so, the restored model has less accuracy loss as the quantization errors will be fed back into the training process.

Based on the above observations and motivations, we propose a novel quantization-based delta compression framework, QD-Compressor, to improve the compression ratio for DNNs while well maintaining the model accuracy. QD-Compressor uses the weighted entropy based local-sensitive quantization to improve the compression ratio and uses the error feedback mechanism to keep the accuracy of the restored model.

4 Design Methodologies

In this section, we describe QD-Compressor design in detail, including quantizing the neural network to better exploit the similarity of the neighboring neural networks, and then calculating the lossless delta data based on the quantized version of network, finally encoding the delta data with lossless encoding schemes.

4.1 Overview of QD-Compressor Framework

The general workflow of QD-Compressor framework is shown in **Figure 2**. To compress a neural network (called target network), we need a reference neural network, which is usually the former version of the target network in training. QD-Compressor will calculate and compress the delta data of two networks for efficient space savings by using the **weighted entropy-based local sensitive quantization algorithm**. More specifically, QD-Compressor consists of four key steps: selection of quantization bit width, network quantizing, delta quantizing, and delta compressing.

1) *Bit width selecting* is to allocate different quantization bit width among layers according to the weighted entropy for achieving greater and more flexible compression. Most network layers might be quantized with low bit width, with just a few critical layers having more bit width.
2) *Network quantizing* is to quantize the floating-point parameters of the neural network for each layer by dynamically feeding the quantization error into the training process by an error feedback mechanism.
3) *Delta calculating* is to calculate the delta data of the quantized parameter between the neighboring neural networks (e.g., target and reference networks). It converts the similarity in values into compressibility in the storage of delta data which will be much more compressible than directly compressing the floating-point parameters in the neural networks.
4) *Delta compressing* is to further reduce the delta data size by using lossless compressors.

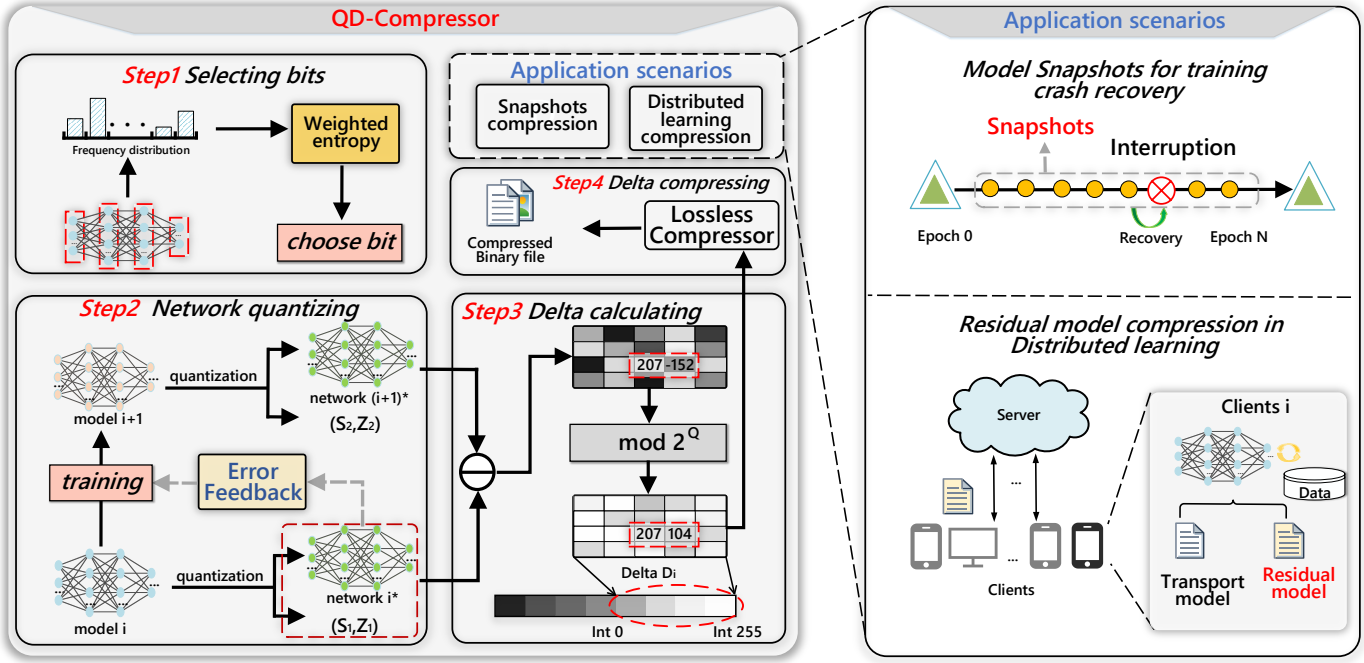


Fig. 2. Overview of QD-Compressor framework for model snapshots and Federated Learning. Model snapshots and stored model are the objects to be compressed. Model i represents the model in the i-th epoch of training.

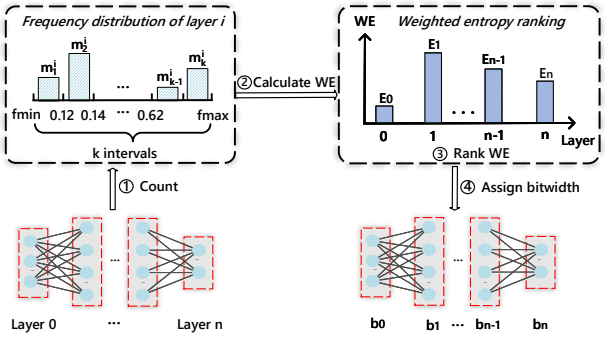


Fig. 3. The workflow of determining the quantization bit width. m_t^i is the number of weights in the t^{th} interval of i^{th} layer. WE denotes the weighted entropy. E_i is the value of the weighted entropy of the i^{th} layer. b_i is the number of bits quantized at the i^{th} layer. Suppose E_0 and E_1 are the minimum and maximum weighted entropy and assigned with the minimum and maximum quantization bits, i.e., b_0 is the minimum bit width and b_1 is the maximum bit width, respectively.

In the remainder of this section, we will discuss these four steps in detail to show how QD-Compressor efficiently compresses the floating-point parameters in neural networks. Simultaneously, we will discuss how QD-Compressor might help decrease storage overhead during the Federated Learning training process.

4.2 Selection of Quantization Bit Width

As introduced in Section 3, due to the parameters distribution differs among layers, we are driven to quantize various layers using different bit width (i.e., the number of bits used for mapping the floating-points to the integers) for achieving a greater and more flexible compression performance. Referred to the state of the arts [40], [41], [42], **Weighted Entropy** is derived from the physics notion of entropy and is designed to take the importance of data into account [41]. In this paper, We employ the weighted entropy as the quantitative criterion to measure the significance of the layer's weight and decide the quantization bit width for different layers.

As shown in the Figure 3, assuming that the ith layer has weight matrix W_i with n_i elements w_j^i, $j \in 1, 2, ..., n_i$. The components of W_i can be considered as independent and identically distributed independently distributed random variables p_i [42]. The weighted entropy of each layer can be approximately calculated as: (1) we uniformly divide the range of weights $[f_{min}, f_{max}]$ into k bins and count the number of weights falling in each interval, where k is a predefined hyper parameter, f_{min} and f_{max} are the minimum and maximum values of weights in this layer, respectively. (2) The frequency is regarded as the approximated probability p_i of t th bin center, $t \in 1, 2, ..., k$. (3) The entropy E_i of i-th layer's weights could be calculated as:

$$E_i = -\sum_{t=1}^{k} p_t^i log(p_t^i) \quad with \quad p_t^i = \frac{m_t^i}{n_i} \tag{1}$$

where m_t^i is the number of weights in the t th bin. After getting the significance values for all layers, they are sorted in order of increasing magnitude. Due to more bit width carry more information, we set long bit width for layers with higher entropy to preserve the representational capability of original model and set less bit width to those with lower entropy (The detailed descriptions about how to select b_{max} and b_{min} are provided in the Subsection 6.2.1). Theoretically, the bit width b_i of the i-th layer can be determined to maintain the same amount of entropy loss to the maximum entropy layer as:

$$b_i = b_{min} + round((b_{max} - b_{min}) \times (\frac{E_i - E_{min}}{E_{max} - E_{min}})) \tag{2}$$

4.3 Network Quantizing

According to subsection 4.2, the bit width B_i of each layer i has been determined, we describe the detailed quantization

scheme in this subsection.

To better utilize the similarity of floating-point parameters whose values are close and avoid mantissa uncertainty, we conduct the linear quantization to map the original floating-point parameter f in floating-point set F to quantized integer value q in integer set Q. More specifically, the quantization operation for a set is mapping the value range $[f_{min}, f_{max}]$ of the floating-point set F to quantized value range $[Q_{min}, Q_{max}]$ of integer set Q. Q_{max} and Q_{min} are the maximum and minimum values that can be represented after quantization (e.g., $Q_{min} = 0, Q_{max} = 255$ for Uint8 quantization). To this end, we use S to scale the size of the value range and then use Z to offset:

$$q = round(\frac{f}{S} + Z) \tag{3}$$

where the constant S is positive real number for scaling the value range, the constant Z is the offset for translating the value range $[\frac{f_{min}}{S}, \frac{f_{max}}{S}]$ to $[Q_{min}, Q_{max}]$, which can be calculated as below:

$$S = \frac{f_{max} - f_{min}}{Q_{max} - Q_{min}} \tag{4}$$

$$Z = Q_{min} - \frac{f_{min}}{S} \tag{5}$$

The key process of quantization is to find the quantization constants of the floating-point set F and the quantized value range $[Q_{min}, Q_{max}]$ of integer set Q. Considering storage efficiency, the quantization level we set in quantization is 2^B (B is the bit width introduced in subsection 4.2). It corresponds to a quantization value range $[0, 2^B]$.

As discussed in Section 3, the value range differs among layers (as shown in **Figure 1**). If we regard all parameters in the DNN as a floating-point set to determine the global quantization constants, the quantization error will be large and degrade the accuracy of the network. Thus, we calculate the quantization constants S and Z for each layer, respectively. More importantly, different quantization constants of each layer facilitate generating more redundant data in the next delta calculating step since the floating-point difference of neighboring versions of the network with different scales may be quantized to the same integer delta data. To prove above conjectured explanation, we further compare the entropy of the quantized data for these two quantization schemes, and the result is shown in **Figure 4**, which illustrates that the local-sensitive quantization can generate low entropy data and thus higher compressibility can be achieved by local-sensitive quantization.

Further more, we also study the order of quantization and delta calculation to explore a higher compressibility. Specifically, we evaluate the compression ratios with different order of quantization and delta calculating in multiple DNNs. **Table 1** suggests that "Quantization before Calculating Delta" always achieves a higher compression ratio than "Quantization after Calculating Delta". That is because similar floating-point between two neighboring model might be quantized to the same integers, leading to more zeros generated. Thus QD-Compressor perform performs quantization ahead of calculating delta.

Algorithm 1: Network Quantizing with Error Feedback Mechanism

Input: Current network: M; quantized network: Q;
restored network $M*$; layer in network M: L;
layer in network Q which corresponding to L: L_Q;
quantize constant: S, Z; quantization bit width: B;
quantization bit width for the Lth layer: b_L;
loss of inference: $loss$; gradient: g;
learning rate: lr;
Output: next epoch network: M_{next}
for L *in* M **do**
  $b_L \leftarrow$ select(B); //Assign bit width for each layer
for L *in* M **do**
  calculate S_L and Z_L in F; //according to b_L
  $L_Q \leftarrow$ quantize (L,S_L,Z_L)
$M* \leftarrow$ restore(Q)
use network $M*$ to calculate $loss$ and gradient g;
$M_{next} = M - lr * g$; //update on the original network M
return M_{next};

4.4 Network Updating

After above quantization algorithm, the network will continue to be trained on the datasets. However, there is inevitable information loss incurred by the lossy quantization, which degrades the model quality. Thus an error feedback mechanism is proposed to solve this problem. More specifically, instead of directly training on the full precise model M, the error feedback mechanism (1) restores the lossy version $M*$ from quantized delta; (2) trains on the lossy version $M*$ and obtains the gradient g; (3) updates the parameters on M (note that rather than $M*$) using the gradient g; In doing so, the quantization error in $M*$ could be introduced into the normal training process in M. The advantage is that the training process will be dynamically corrected in each training round, thus the accuracy of restored model is well maintained.

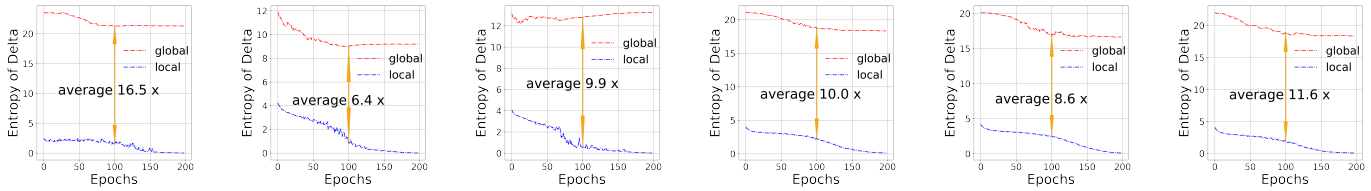
In summary, we (1) determine the quantization bit width according to the weighted entropy for different layer, (2) separately select the constants S and Z according to the range of the parameters for each different layer and perform the linear quantization. (3) The quantization error is fed back into the training process during the training. The whole process is described in **Algorithm 1**.

4.5 Delta Calculating

We have obtained the quantized version of two neighboring networks in the 4.4. Here we can take full advantage of the similarity between these two quantized networks to achieve a high compression ratio. Before calculating the delta data, we quantitatively analyze the similarity between two quantized versions of neighboring networks.

More specifically, given two numerical sequences with the same length L_1 and L_2, we employ two metric to measure the similarity of L_1 and L_2: Manhattan distance mean**(MDM)** and correlation coefficient**(CorrCoef)**, which are calculated according to **Equation (6)** and **Equation (7)**.

$$MDM(L_1, L_2) = \frac{1}{n}\sum_{i=1}^{n}|L_1[i] - L_2[i]| \tag{6}$$

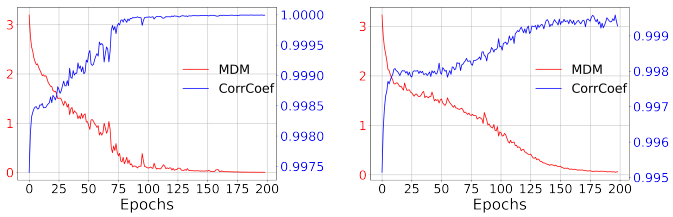


(a) VGG-16 (b) ResNet-18 (c) GoogleNet (d) MobileNet (e) ShuffleNet(Size=1) (f) ShuffleNet(Size=2)

Fig. 4. Entropy of delta data comparison between global quantization and local-sensitive quantization.

TABLE 1
Compression ratio comparison between global and local-sensitive quantization before and after calculating delta

Compressor	VGG-16	ResNet-18	GoogleNet	MobileNet	ShuffleNet(Size=1)	ShuffleNet(Size=2)
Global Quantization	1260MB (8.92)	800MB (10.41)	451MB (10.21)	172MB (10.05)	99MB (9.61)	349MB (11.45)
Local-Sensitive Quantization after Calculating Delta	541MB (20.77)	552MB (15.46)	259MB (18.21)	156MB (11.41)	91MB (10.73)	306MB (13.34)
Local-Sensitive Quantization before Calculating Delta	465MB (24.17)	443MB (19.26)	211MB (22.35)	119MB (14.96)	69MB (14.15)	230MB (17.82)



(a) GoogleNet (b) MobileNet

Fig. 5. MDM and CorrCoef of GoogleNet and MobileNet.

$$CorrCoef(L_1, L_2) = \frac{cov(L_1, L_2)}{\sigma_{L_1} * \sigma_{L_2}} = \frac{\sum_{i=1}^{n}(L_1[i] - \overline{L_1})(L_2[i] - \overline{L_2})}{\sqrt{\sum_{i=1}^{n}(L_1[i] - \overline{L_1})^2} * \sqrt{\sum_{i=1}^{n}(L_2[i] - \overline{L_2})^2}} \quad (7)$$

MDM is calculated to measure the sum of the absolute wheelbase of two points in the standard coordinate systems. Therefore, we regard the parameters sequence in the network as a vector to calculate the Manhattan distance and divide it by the length to obtain the average value's differential of each parameter (i.e., Manhattan distance mean **(MDM)**). The smaller the MDM value is, the more similar the sequences are.

CorrCoef [43] is calculated to measure the coefficient of correlation between two variables which is widely used in statistics. The closer the correlation coefficient vale is to 1, the greater the similarity degree of two sequences will be.

Figure 5 shows MDM and CorrCoef of quantized parameters' sequences in GoogleNet and MobileNet. And there are total 199 pairs of neighboring versions in 200 training epochs. MDM (red curve) is the average value of the difference between the quantized values of each parameter in the neighboring versions. The value ranges of the red curve from 0 to 3 means the difference between the corresponding parameter in neighboring versions is very small. CorrCoef (blue curve) represents the coefficient between two quantized parameters sequence of the neighboring versions. The value range of the blue curve is all above 0.99 and gradually approaching 1, which proves that the quantized parameters

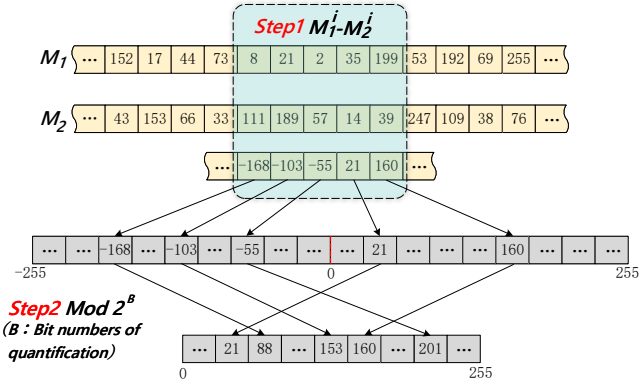


Fig. 6. Diagram of calculating the delta of quantized parameters in QD-Compressor.

in neighboring versions are very similar. These observations are the important supports for the next compression (see Subsection 4.6).

We denote the quantized model parameters of the latest version as M_1, and the corresponding quantized model parameters of the previous version as M_2, the quantized delta data D_i can be calculated as M_1^i-M_2^i. However, in B bits quantization, the value range of M_1^i and M_2^i is $[Q_{min}, Q_{max}]$, so the value range of delta data D_i will be $[Q_{min} - Q_{max}, Q_{max} - Q_{min}]$ which means the delta value needs one additional bit storage space than the quantized parameters. Thus we regard the quantized value range as a directional loop range.

Figure 6 illustrates an example of calculating delta data in QD-Compressor. When $M_1^i > M_2^i$, the delta data $D_i = M_1^i - M_2^i$. Otherwise $D_i = (Q_{max} - M_2^i) + 1 + (M_1^i - Q_{min})$, where $Q_{max} - Q_{min} + 1 = 2^B$. Therefore, $D_i = M_1^i - M_2^i + 2^B$ when $M_1^i < M_2^i$. As a whole, the value range of delta data D_i is from zero to $Q_{max} - Q_{min}$, can be calculated as below:

$$D_i = (M_1^i - M_2^i) \mod 2^B \quad (8)$$

Because there exists a high similarity of quantized parameters, the delta data will have great redundancy for further compression.

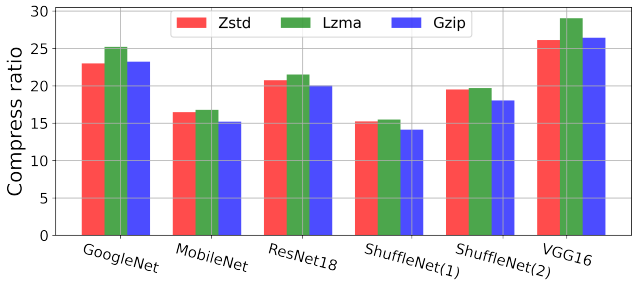


Fig. 7. Compression ratio of using Zstd, Lzma, and Gzip compressing the delta data of six networks after quantization.

Algorithm 2: Efficient Distributed/Federated Learning with the Residual Accumulation via QD-Compressor.

Input: initial model parameters W; epochs: T; number of the clients set C; compressed residual accumulation QR (initial to zero);
Output: model parameters W after training
for t *in* T **do**
  for i *in* C *parallel* **do**
    $R_t^i \leftarrow$ QD-Decompress(QR_t^i);//Decompress R_t^i
    ΔW_t^i = SGD(W_t^i)-W_t^i; //Train
    $\Delta W_t^i \leftarrow \Delta W_t^i + R_t^i$; //Accumulate with R_t^i
    $msg_t^i \leftarrow$ Topk(ΔW_t^i); //Filter parameters
    $R_{t+1}^i \leftarrow \Delta W_t^i$ - msg_t^i; //Update R_t^i
    $QR_{t+1}^i \leftarrow$ QD-Compress(R_{t+1}^i); //Compress R_t^i
    uploads msg_t^i to Server;
  Server S does:
    gather msgs from clients;
    $W_{t+1} \leftarrow$ FedAverage(msgs); //Average model
    sends W_{t+1} to the chosen clients;

4.6 Delta Compressing

When the quantized delta data is obtained, QD-Compressor will compress these delta data using lossless compressors to achieve a high compression ratio. Here we study three typical lossless compressors Zstd, LZMA, and GZip, to evaluate the final compression ratio on the delta data.

Figure 7 shows the compression ratio of Zstd, LZMA, and GZip in our framework. Through comparing the compression performance of these mainstream lossless compressors, we can observe that LZMA achieves the highest compression ratio (as shown in **Figure 7**). Thus LZMA is selected as the lossless compressor to compress the quantized delta data into a compressed binary file.

4.7 Model Decompressing

When we need to decompress the neural network of the current version: (1) We decompress the compressed binary file to the delta data with LZMA. (2) The decompressed delta data is added to the quantized network of the previous version. (3) We use the quantized constants S and Z of each layer stored in local before, to restore the floating-point version R^* of the network according to **Equation** 9.

$$R^* = S \times (Q_{min} - Z) \tag{9}$$

In order to decompress the network of a specific version (e.g., snapshots restore), we can execute the above 3 steps recursively until we obtain the target network version.

However, the remaining parameters have the same size as the full model [14], [15], [44]. As a result, additional storage space is required to accommodate the residual model.

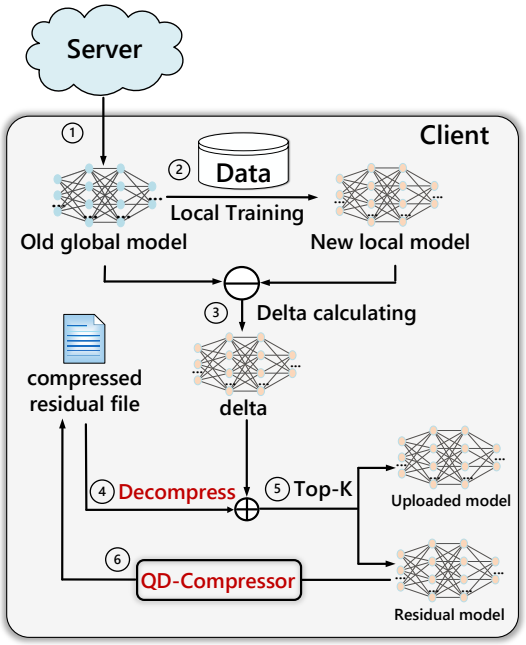


Fig. 8. Distributed/Federated Learning with the residual accumulation via QD-Compressor.

TABLE 2
Parameter number and FLOPs of popular DNNs

Network	Para(M)	FLOPs(G)	Network	Paras(M)	FLOPs(G)
densnet121	7.98	2.90	VGG11	132.86	7.74
densnet169	14.15	3.44	VGG16	138.36	15.61
mobilenet_v2	3.50	0.33	resnet18	11.69	1.82
inception_v3	27.16	5.75	resnet34	21.80	3.68

For the resource-constrained edge devices, this additional storage can not be ignored.

4.8 Complexity Analysis

As discussed before, our proposed QD-Compressor consists of four parts: selection of quantization bits, quantizing the network, calculating the delta data, and using a lossless compressor to compress the delta data.

In the stage of selection of quantization bits, we need to calculate the entropy of weights of each layers. The time complexity of determining the quantized constants is $O(N)$, where N is the total number of parameters in the whole neural network. In the stage of network quantizing, We need to traverse parameters for each layer to determine the quantized scaling constants S and offset Z. The time complexity is the same as the previous stage. After obtaining the quantization constants S and Z, the network can be quantized and the delta data can be calculated at the same time. The operation of this process is to quantize each parameter by using the linear transformation in **Equation (3)** and calculating the subtraction of the difference by **Equation (8)**. The time complexity of quantizing the network and calculate the delta data is also $O(N)$. So, the time complexity of these two stages is $O(N)$. **Table 2** shows the number of parameters and operations in different network. From this table, the number of operations is much bigger than parameters which means $O(N) << O(F)$ (F is the FLOPs of the

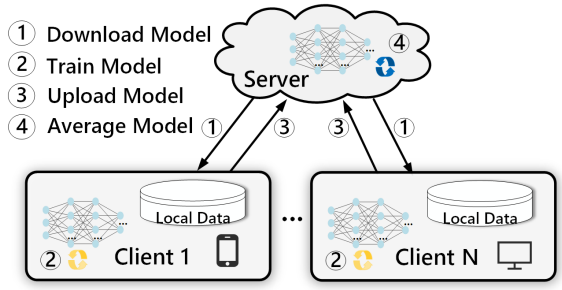


Fig. 9. The general workflow of Federated Learning. The server is responsible for sending and aggregating the clients' models, and the client trains the model based on its own local data.

network). At the same time, in each training epoch, the total time complexity is $O(d * F)$ (d is the amount of data in the data set). Thus, we can get: $O(N) << O(F) << O(d * F)$, which means **the complexity of our quantizing method is much smaller than the training process**.

The last stage of lossless compression can be parallel to the training of the neural network after calculating the delta data. At the same time, Delta-DNN [21] also points out that the cost of lossless compression is very small compared to the time spent on training and testing the network. In summary, the computation overhead of additional operations in QD-Compressor is lightweight compared to the normal training process.

5 Realistic Application Scenarios

In this section, two main applications of our proposed QD-Compressor are described.

5.1 Top-K Sparsification-Based Federated Learning

As mentioned in Section 1, in the resource-constrained federated context, the main bottlenecks are the limited bandwidth and storage in the edge nodes (clients) (e.g., smartphones, IoT devices). To reduce the communication cost between client and server, the **Top-K Sparsification** technique is always used compress the transmitted model parameters. Broadly speaking, when one client participates in the learning at round t, it selects and sends top-k elements of the update with largest magnitude to the server. To maintain the model quality, the remainder of unselected elements (a.k.a., residual model) will be "stashed" in the local disk [45]. In general, the technique of storing the residual model locally, known as **Residual Accumulation**, provides the important advantage of minimizing update information loss in FL. (it may only become outdated or "stale") [14], [15], [45]. When this client participates in FL at round $t+\tau$, the remaining parameters will be added back to the relevant trained local updates (as illustrated in **Algorithm 2**).

Due to residual model can be regarded as a delta of two neighboring local models, which is appropriate to be compressed by our proposed QD-Compressor as follows: (1)the client trains the local model and obtains the updates in the round t. (2) A small part of the elements of the update will be selected by the Top-K sparsification and uploaded to the server. (3) The residual model is compressed by QD-Compressor (note that this step can run simultaneously with the communication process, thus doesn't incur additional time cost to Federated Learning). (4) When the same client participates in the round $t+1$, we decompress the compressed binary file of the residual model. (5) After the local training, the decompressed residual model is added back to the new updates, and conduct the sparsification again.

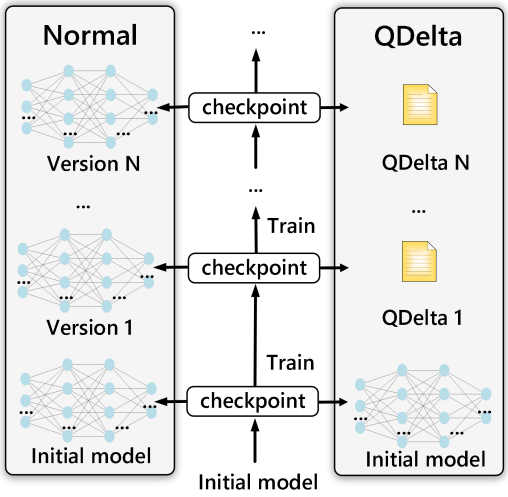


Fig. 10. Process for saving model snapshots. QDelta is the compressed delta data of each version.

Our evaluation results suggest that our proposed QD-Compressor can achieve 11×-15× compression ratio for the residual model without degrading the model training performance, which will be illustrated in Subsection 6.3.

5.2 Snapshots Recovery for The Training Crash

During the DNNs training, the infrastructure and process failures are widespread in large data clusters, with an MTBF of 4-22 hourss [46], [47], [48], [49], [50], [51], [52]. In a Microsoft cluster, the facility failures, node breakdowns, and software faults take an average of 45 minutes to occur throughout the neural network training. Thus the snapshots of historical versions are important intermediate results. On another hand, the snapshots also give detailed information about intermediate states, which can be used to improve the scientific analysis and ensemble learning [10].

However, because the size of the snapshot is the same as the full model, the storage costs of multiple snapshots becomes a storage bottleneck for resource-constrained devices. Fortunately, due to the magnitude of the parameter change is relatively small between the neighboring version of the two snapshots, the difference between two neighboring models (i.e., delta) is small. As a result, we can use QD-Compressor to achieve a high compression ratio on this delta data. In doing so, the storage costs of multiple snapshots are significantly reduced by only storing one full model of $1st$ round, and the compressed delta models of all subsequent rounds.

In the snapshot recovery stage, we can recover the model of $i-th$ round by iteratively decompressing the compressed delta and adding back to the full model of $1st$ round. Due to QD-Compressor being a lossy compression approach, we need to take a few rounds of fine-tuning for the recovered model to reach the target accuracy (i.e., the original accuracy of $i-th$ model before the training crash).

TABLE 3
Inference accuracy of full-precision with 4-bit and 8-bit

	VGG-16	ResNet18	GoogleNet	MobileNet	ShuffleNet(size=1)	ShuffleNet(size=2)
full-precision accuracy	92.00%	93.65%	93.68%	92.38%	90.44%	91.34%
4-bit accuracy	87.76% (–4.24%)	92.35% (–1.30%)	80.92% (–12.76%)	91.67% (–0.71%)	89.76% (–0.68%)	N/A
8-bit accuracy	91.92% (–0.08%)	93.89% (+0.24%)	93.72% (+ 0.04%)	92.36% (–0.02%)	90.34% (–0.10%)	91.53% (+0.19%)

In the permanent storing of neural work, snapshots must be saved. However, because to the vast size of the neural network, the storage overhead of preserving the historical version takes a significant amount of space. The goal of decreasing storage overhead may be accomplished in this scenario by just preserving the delta data of each version (as shown in **Figure 10**).

TABLE 4
Compression ratio comparison between 8-bit and 16-bit quantization-based delta compression in MobileNet and VGG-16

Network		Origin	8-bit	16-bit
MobileNet	Total size	1.738G	106M	510M
	Compression ratio	N/A	16.79	3.49
VGG-16	Total size	10.974G	387M	2777.54M
	Compression ratio	N/A	29.04	4.05

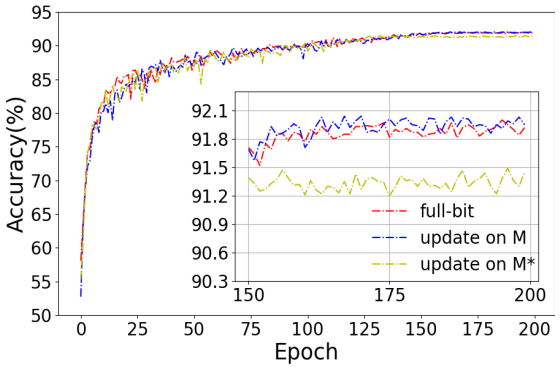


Fig. 11. VGG-16 inference accuracy with full-precision, updating on M and updating on M* quantization.

6 Performance Evaluation

In this section, we (1) study the metrics of test accuracy and compression ratio on different configuration of QD-Compressor's hyper-parameter (i.e., quantization bits); (2) compare QD-Compressor, and Zstd, SZ, Delta-DNN, LC-Checkpoint compression approaches on the metric of compression ratio on multiple typical DNNs; (3) evaluate the performance of QD-Compressor in the two mainstream application: *(i)* snapshot recovery from the training crash and *(ii)* local accumulation compression in Federated Learning.

6.1 Experimental Setup

We conduct our experiments on an Ubuntu server with an Intel Xeon 6154 (with 32GB of memory) and two RTX3090. Our experiments involve two parts: Basic performance evaluation and realistic application evaluation. The basic performance evaluations include ablation experiments and compression ratio comparisons. The realistic application evaluations include snapshot recovery from training crash from and Federated Learning residual compression.

(i) For the Basic performance evaluation and **snapshot recovery from training crash** experiments, the model training is performed on the PyTorch deep learning framework [53] with six typical DNNs: VGG-16 [9], ResNet-18 [54], GoogleNet [2], MobileNet [55], ShuffleNet [56]. We train each DNNs on CIFAR-10 dataset [57] with 200 epochs. The optimizer used in model training are SGD with learning rate=0.01, momentum=0.9, and weight decay=5e-4.

(ii) For the **Federated Learning residual compression** experiments, the model training and communication are performed by the PyTorch deep learning framework and the python socket communication framework. The Federated Learning is performed on ten clients and one server at the cloud environment with 1Gbps bandwidth. Each each client first trains the local model on VGG-16 and Cifar10 for 1 epoch, and then compresses the updates using Top-K sparsification. The selected elements will be uploaded to the server, and the remaining unselected residuals will be compressed by different compressor. The optimizer used in model training are Adam with learning rate=0.001.

6.2 Basic Performance Evaluation

6.2.1 Selection of Quantization Bits

In our proposed QD-Compressor framework, the hyper-parameters are the maximum bit width b_{max} and the minimum bit width b_{min} for each layer of the floating-point network. Considering the storage efficiency, the bit width are set to multiple or a fraction of a byte which corresponds to three options: 4-bit, 8-bit, and 16-bit.

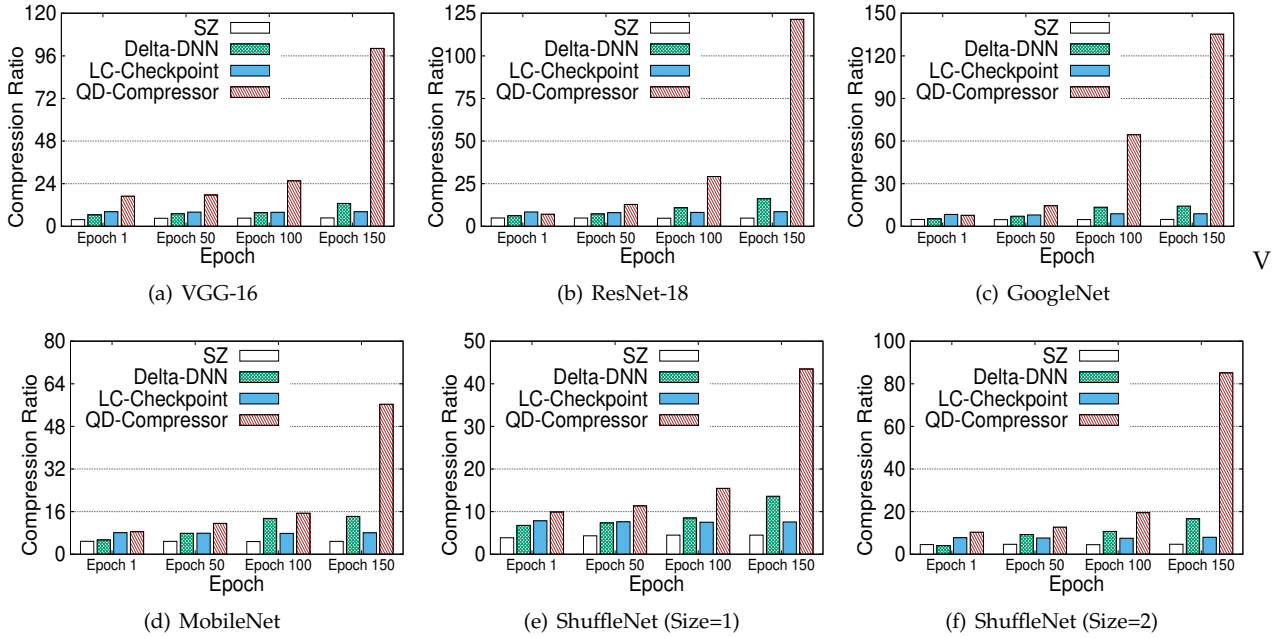
In our algorithm, a necessary premise is that the restored model inference accuracy can not be significantly degraded after quantization. Under this premise, we select the hyper-parameters by achieving a high compression ratio while well maintaining the model's inference accuracy. Therefore, we training the network by QD-Compressor on different configurations of **fixed B bit** (e.g., 4-bit, 8-bit), and observe the inference accuracy.

As shown in **Table 3**, the convergence of the model is significantly affected when using 4-bit as the quantized bits of the whole network. 'N/A' of ShuffleNet (size=2) in **Table 3** means network can not converge. Although the accuracy of VGG-16 and GoogleNet decreases a lot and the training process is unstable, the smaller network model like MobileNet and ShuffleNet (size=1) has less accuracy loss. As a result, 4-bit satisfies with our premise and we don't choose a lower number of bits than 4-bit as the minimum bit width b_{min}.

Meanwhile, when we increase the bit size to 8-bit and 16-bit, respectively, there is a nearly overlapped accuracy

TABLE 5
Compression ratio of Zstd, SZ, Delta-DNN, LC-Checkpoint, QD-Compressor (fixed 8 bit) and QD-Compressor (varied bit width from 4 to 8)

Network	Model Size	Total Size	Compress Size (Compression Ratio)					
			Zstd	SZ	Delta-DNN	LC	QD (8bits)	QD (4-8bits)
VGG-16	56.19MB	10.974GB	10.17GB(1.079)	2.264GB(4.643)	1.231GB(8.92)	1.35GB(8.10)	387MB(29.04)	281MB(39.99)
Resent-18	42.66MB	8.333GB	7.69GB(1.084)	1.750GB(4.763)	800.9MB(10.41)	1.01GB(8.24)	398MB(21.51)	305MB(27.98)
GoogleNet	23.58MB	4.606GB	4.26GB(1.082)	973.5MB(4.731)	451.2MB(10.21)	550.97MB(8.56)	187MB(25.22)	142MB(33.22)
MobileNet	8.9MB	1.738GB	1.61GB(1.080)	375.2MB(4.631)	172.9MB(10.05)	223.07MB(7.98)	106MB(16.79)	82MB(21.70)
ShuffleNet(Size=1)	4.88MB	0.977GB	0.908GB(1.076)	213.4MB(4.469)	99.2MB(9.61)	128.28MB(7.61)	63MB(15.50)	48MB(20.30)
ShuffleNet(Size=2)	20.49MB	4.002GB	3.70GB(1.083)	871.9MB(4.590)	349.5MB(11.45)	529.1MB(7.75)	208MB(19.70)	162MB(25.29)



(a) VGG-16 (b) ResNet-18 (c) GoogleNet

(d) MobileNet (e) ShuffleNet (Size=1) (f) ShuffleNet (Size=2)

Fig. 12. Compression ratio in different epochs of training process using SZ, Delta-DNN, LC-Checkpoint and QD-Compressor.

curve with full-precision baseline as shown in **Table 3**. Thus we can conclude that there is no significant accuracy degradation when the quantization bits number is greater than or equal to 8-bit. However, the experimental result in **Table 4** shows that the compression ratio of 16-bit is far lower than that of 8-bit. Thus, 8-bit is set to the maximum bit width b_{max} for QD-Compressor to maximize the compression ratio.

6.2.2 Evaluation of Error Feedback Mechanism

In this Subsection, we evaluate the impact of implementing an error feedback mechanism on the model accuracy, the non-compression full-precision model training is employed as the baseline.

As described in Subsection 4.4, the essence of error feedback is to introduce the quantization error into the training process by updating the raw model M using the gradients calculated on the lossy model $M*$ (as described in **Algorithm 1**). To prove the advantage of error feedback mechanism, we compare the error feedback mechanism with the other updating method: updating the lossy model $M*$ using the gradients calculated on the $M*$, and using the $M*$ as the new model to train. As shown in **Figure 11**, the accuracy of baseline and the method of updating on M are nearly overlapped, which suggests that they have almost the same convergence performance. However, the highest accuracy curve of the method of updating on $M*$ is reduced by nearly 0.5%, which is lower than the method of updating on the raw model M. This proves that our error feedback mechanism (i.e., updating on the inverse quantized model M) can effectively feed the quantization error to the normal DNNs training process.

To further prove the efficiency and universality of our error feedback mechanism. We quantize six popular DNNs with 8 bits and compare the test accuracy under above two updating methods (i.e., updating on M or $M*$). **Table 6** shows that QD-Compressor's error feedback can achieve the comparable test accuracy with full-precision baseline in different popular networks and better than updating on $M*$. For each network, the difference between the final test accuracy (i.e., convergence accuracy) is even smaller than the fluctuation in the normal training process of the network, which proves the efficiency and universality of our proposed error feedback mechanism.

6.2.3 Evaluation of Compression Ratio

In this subsection, we evaluate the total storage overheads on six popular DNNs for different compression methods. Note that the total storage overheads are calculated by the sum of compressed network sizes of all rounds. **Table 5**

TABLE 6
The test accuracy (%) with full-precision, updating on M and updating on M* QD-Compressor (QD) for different DNNs

Network	Full Precision	QD (on M*)	QD (on M)
VGG-16	92.00	91.49(-0.51)	91.92(-0.08)
ResNet-18	93.65	93.39(-0.26)	93.72(+0.07)
GoogleNet	93.68	93.51(-0.17)	93.72(+0.04)
MobileNet	92.38	92.05(-0.33)	92.36(-0.02)
ShuffleNet(Size=1)	90.44	89.93(-0.51)	90.37(-0.07)
ShuffleNet(Size=2)	91.34	91.21(-0.13)	91.53(+0.19)

TABLE 7
Model size and compression ratio of LC-Checkpoint and QD-Compressor in Federated Learning

Network	Model Size	Compressd Size (Compression Ratio)	
		LC-Checkpoint	QD-Compressor
VGG-16	59MB	5.83MB(9.64)	5.54MB(10.65)
ResNet-18	44MB	3.06MB(14.38)	2.71MB(16.22)
ResNet-34	84MB	6.72MB(12.49)	5.61MB(14.96)

shows that QD-Compressor achieves the highest compression ratio among all compared methods.

In order to further study the compression efficiency of QD-Compressor in the process of DNNs training in detail, we further studies the compression ratio of the six networks under four compression methods **at different training epochs** in **Figure 12**. This is because they target the network itself but not the connection between the versions during the training process. The compression ratio of Zstd is just near 1.1 because floating-point numbers are hard to be compressed directly due to their mantissa uncertainty. In the Delta-DNN method, the compression ratio will gradually increase with the network training process as the network parameters change smaller and smaller in the end. In our method, the compression ratio at the beginning of training exceeds that of other methods. Additionally, the compression ratio is increasing during the training as the similarity between the neighboring versions increases. Furthermore, the weight entropy-based quantization strategy achieves higher compression ratio than the fixed 8-bit quantization without degrading the test accuracy. In the next experiments, the default configuration of bit-width of QD-Compressor is set to QD (4-8).

6.3 Realistic Application Evaluations

In this subsection, we compare the performances (e.g., compression ratio, model quality, and compression costs) with the state-of-the-art snapshot compressor LC-Checkpoint on two realistic application scenarios: Residual compression for Federated Learning and Snapshot recovery from the training crash.

6.3.1 Residual compression for federated learning

As introduced in 6.1, we jointly train the typical VGG-16 and ResNet on CIFAR-10 by the Federated Learning context (ten clients and one server are communicated with 1Gbps bandwidth). To reduce the communication costs, the client only uploads a small part of updates to the server (e.g., Top-0.01 sparsification means upload 1% of parameters) for each Federated Learning round. To keep the model quality, the remaining of updates will be stored locally and added back to in the next round (i.e., residual accumulation). In this subsection, we use QD-Compressor to reduce the storage cost for the residual model.

TABLE 8
Compression time (CT) and decompression time (DT) of LC-Checkpoint and QD-Compressor in Federated Learning

Network	Training Time	LC-Checkpoint		QD-Compressor	
		CT	DT	CT	DT
VGG-16	24.37s	8.60s	14.43s	0.21s	0.22s
ResNet-18	24.67s	6.11s	8.01s	0.21s	0.21s
ResNet-34	32.46s	13.55s	17.14s	0.23s	0.25s

TABLE 9
Recovery epochs in different periods

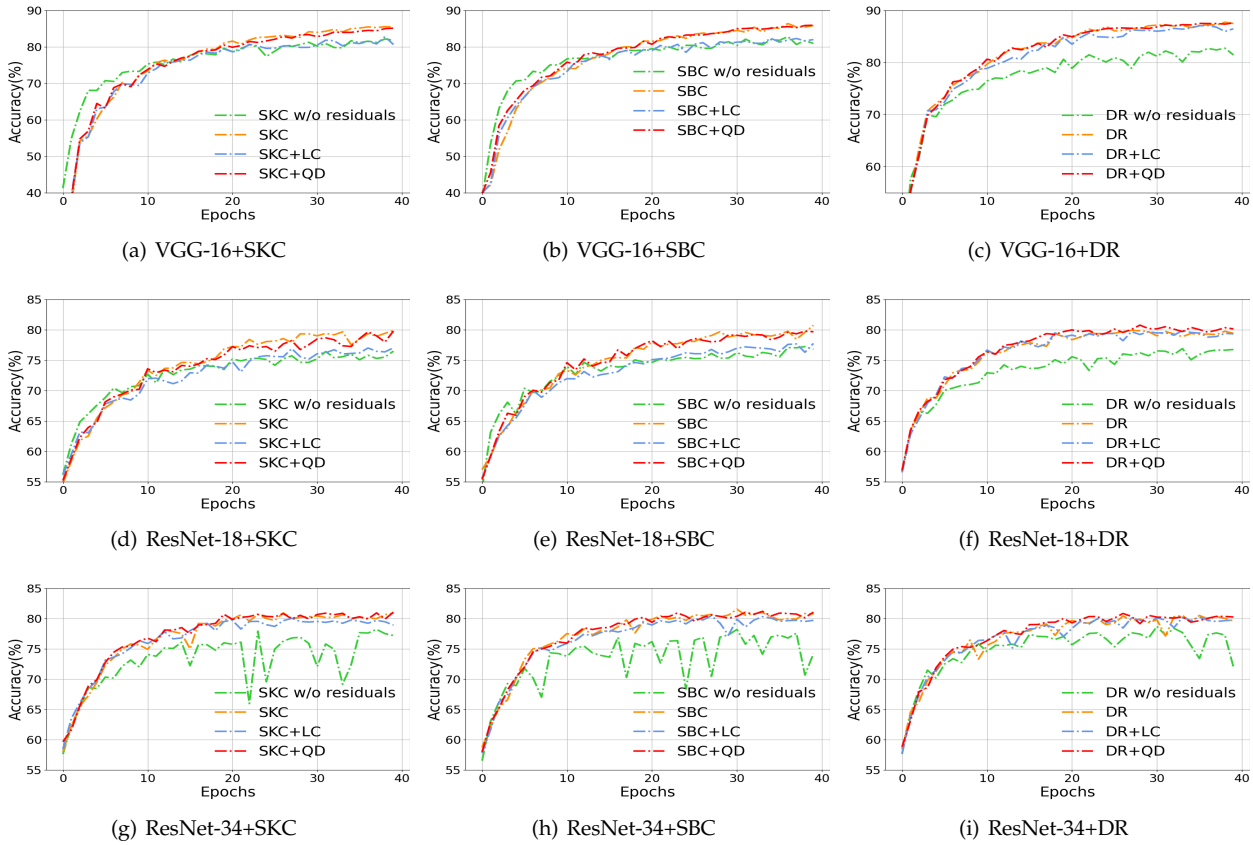
Network	Recovery epochs					
	1-50	51-100	101-150	151-200	1-100	101-200
VGG-16	1/3	2/2	5/2	3/2	1/17	1/6
ResNet-18	1/2	1/3	3/3	1/2	2/32	1/14
GoogleNet	2/3	4/7	1/6	2/3	1/27	1/15
MobileNet	3/8	1/5	1/4	1/1	2/16	1/12
ShuffleNet (Size=1)	2/5	2/4	2/5	2/2	3/16	1/10
ShuffleNet (Size=2)	2/3	2/12	2/4	2/6	4/9	2/7

More specifically, we deploy QD-Compressor on three recent Top-K sparsification-based Federated Learning compression algorithms (i.e., SBC [44], SKC [58], and DeepReduce [59]). We evaluate the test accuracy or residual compression ratio on three settings: (1) without residual accumulation, (2) with residual accumulation, (3) with residual accumulation compressed by QD-Compressor. **Figure 13** shows that when we don't use the residual accumulation (i.e., dropping the elements not uploaded to the server) in the Federated Learning, the training performance (i.e., test accuracy of the resulting model) will be degraded significantly, which proves that the residual accumulation is essential to keep the Federated Learning model quality. In order to reduce the storage cost of the residual model, we employ QD-Compressor to compress the local accumulation, the model accuracy curve is almost identical to that without quantization, and even better. It is because that moderate noise doesn't harm the convergence and even improves the generalization. Furthermore, **Table 7** shows the average model size (MB) after compression and the corresponding compression ratio of the local residual parameters during the target communication rounds. The experimental results show that our proposed QD-Compressor efficiently reduces the size of residual accumulation by the factor of 11×-15×. Additionally, since the only cost of QD-Compressor introduced in the Federated Learning process is the decompressing time, we also evaluate the decompressing time and the local training time (in seconds) in **Table 8**. We learn that the cost of QD-Compressor only takes a very small proportion of the whole Federated Learning process.

Summarily, our proposed QD-Compressor efficiently reduces the storage cost of Federated Learning client without degrading the model quality.

6.3.2 Snapshot recovery from the training crash

In this subsection, we compare QD-Compressor to a concurrent snapshot compressor LC-Checkpoint in the training



(a) VGG-16+SKC (b) VGG-16+SBC (c) VGG-16+DR

(d) ResNet-18+SKC (e) ResNet-18+SBC (f) ResNet-18+DR

(g) ResNet-34+SKC (h) ResNet-34+SBC (i) ResNet-34+DR

Fig. 13. Test accuracy on different DNNs with different methods in Federated Learning.

crash two aspects: (1) the fine-tuning rounds from the specific crash nodes to compare the accuracy of the compressed snapshot. (2) the size of all compressed snapshots to compare the compression ratio.

Table 9 suggests that our proposed QD-Compressor spends less fine-tuning rounds to recover restored model to the target accuracy before the training crash, this superiority is more obvious when the large period. Essentially, since the quantization error will be feedback to the training process in QD-Compressor, the gap between a compressed snapshot and the original model is narrow, and fewer quantization errors are accumulated. Additionally, **Table 5** shows that the storage costs of QD-Compressor for the snapshot are less than LC-Checkpoint. That is because the delta data in QD-Compressor has higher compressibility, leading to a high compression ratio.

Finally, the time costs (in seconds) breakdown of QD-Compressor are provided. **Table 10** suggests that the quantization operation of the error feedback process and snapshot decompression operations only take a small share of time cost during the whole training, which means our proposed QD-Compressor possesses the lightweight property in the application of snapshot recovery from training crash.

In summary, compared with the state of the arts, QD-Compressor achieves a higher compression ratio and faster snapshot recovery from the training crash with small costs.

TABLE 10
The time costs breakdown of QD-Compressor in the snapshot recovery from training crash scenario

Networks	Training	Store	Quan	LZMA Comp.	LZMA Decomp.	Restore
VGG-16	9.15s	0.14s	0.02s	10.66s	0.17s	0.03s
ResNet-18	15.56s	0.15s	0.03s	5.37s	0.43s	0.03s
GoogleNet	62.01s	0.13s	0.06s	2.35s	0.23s	0.08s
MobileNet	18.51s	0.06s	0.05s	0.70s	0.09s	0.06s
ShuffleNet (Size=1)	12.2s	0.06s	0.05s	0.33s	0.07s	0.05s
ShuffleNet (Size=2)	18.92s	0.07s	0.04s	1.82s	0.19s	0.05s

7 Conclusion

In this paper, aiming at the phenomenon that a lot of redundancy information exists in neighboring versions of DNNs during the training, we propose a novel quantization-based delta compressor called QD-Compressor. The weighted entropy-based local-sensitive quantization technique with error feedback mechanism of QD-Compressor significantly reduce the storage cost for DNNs without degrading the model quality. Experimental results on two realistic applications and multiple popular DNNs suggest that compared with the state of the arts, QD-Compressor not only achieves 7×-40× higher compression ratio in the model snapshots compression, but also obtains the 11×-15× storage costs for the Top-K sparsification-based Federated Learning clients.

8 ACKNOWLEDGEMENT

This research was partly supported by the National Key-Research and Development Program of China under Grant No. 2020YFB2104003, the National Natural Science Foundation of China under Grant no. 61972441; Shenzhen Science and Technology Innovation Program under Grant no. RCYX20210609104510007, no. JCYJ20200109113427092, and no. GXWD20201230155427003-20200821172511002; Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (2022B1212010005). The research was also partly supported by the U.S. National Science Foundation Grants OAC-2034169/2303820 and OAC-2042084/2303064.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[2] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9.
[3] Y. Wang, W. Gan, J. Yang *et al.*, "Dynamic curriculum learning for imbalanced data classification," in *Proc. ICCV*, 2019, pp. 5016–5025.
[4] R. B. Girshick, J. Donahue, T. Darrell *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. CVPR*, 2014, pp. 580–587.
[5] S. Ren, K. He, R. B. Girshick *et al.*, "Faster R-CNN: towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015, pp. 91–99.
[6] B. Zhuang, C. Shen, M. Tan *et al.*, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proc. CVPR*, 2019, pp. 413–422.
[7] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Proc. CVPR*, 2013, pp. 3476–3483.
[8] X. Zhu, Z. Lei, X. Liu *et al.*, "Face alignment across large poses: A 3d solution," in *Proc. CVPR*, 2016, pp. 146–155.
[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015.
[10] B. Nicolae, J. Li, J. M. Wozniak *et al.*, "Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 172–181.
[11] H. Chai, Z. Yin, Y. Ding *et al.*, "A model-agnostic approach to mitigate gradient interference for multi-task learning," *IEEE Transactions on Cybernetics*, pp. 1–14, 2022.
[12] G. Huang, Y. Li, G. Pleiss *et al.*, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.
[13] W. Zhang, J. Jiang, Y. Shao *et al.*, "Snapshot boosting: a fast ensemble framework for deep neural networks," *Science China Information Sciences*, vol. 63, no. 1, pp. 1–12, 2020.
[14] F. Sattler, S. Wiedemann, K.-R. Müller *et al.*, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.
[15] Y. Lin, S. Han, H. Mao *et al.*, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. ICLR*, 2018.
[16] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proc. NIPS*, 2016, pp. 1379–1387.
[17] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.
[18] B. Jacob, S. Kligys, B. Chen *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. CVPR*, 2018, pp. 2704–2713.
[19] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.
[20] M. Rastegari, V. Ordonez, J. Redmon *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. ECCV*, ser. Lecture Notes in Computer Science, vol. 9908, 2016, pp. 525–542.
[21] Z. Hu, X. Zou, W. Xia *et al.*, "Delta-dnn: Efficiently compressing deep neural networks via exploiting floats similarity," in *Proc. ICPP*, 2020, pp. 40:1–40:12.
[22] Y. Chen, Z. Liu, B. Ren *et al.*, "On efficient constructions of checkpoints," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 1627–1636.
[23] P. Deutsch *et al.*, "Gzip file format specification version 4.3," RFC 1952, May, Tech. Rep., 1996.
[24] F. Zhang, J. Zhai, X. Shen *et al.*, "Tadoc: Text analytics directly on compression," *The VLDB Journal*, vol. 30, no. 2, pp. 163–188, 2021.
[25] ——, "Poclib: a high-performance framework for enabling near orthogonal processing on compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 459–475, 2021.
[26] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.
[27] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, 1977.
[28] X. Zou, T. Lu, W. Xia *et al.*, "Performance optimization for relative-error-bounded lossy compression on scientific data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1665–1680, 2020.
[29] M. Dutch, "Understanding data deduplication ratios," in *SNIA Data Management Forum*, vol. 7, 2008.
[30] T. Suel and et al, "Algorithms for delta compression and remote file synchronization," *Lossless Compression Handbook*, 2002.
[31] W. Xia, H. Jiang, D. Feng *et al.*, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.
[32] P. Dong, S. Wang, W. Niu *et al.*, "Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[33] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IPDPS*, 2016, pp. 730–739.
[34] X. Liang, S. Di, D. Tao *et al.*, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *Proc. IEEE BigData*, 2018, pp. 438–447.
[35] D. Tao, S. Di, Z. Chen *et al.*, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *Proc. IPDPS*, 2017, pp. 1129–1139.
[36] J. Luo, J. Wu, W. Lin *et al.*, "Thinet: A filter level pruning method for deep neural network compression," in *Proc. ICCV*, 2017, pp. 5068–5076.
[37] Z. Liu, M. Sun, T. Zhou *et al.*, "Rethinking the value of network pruning," in *Proc. ICLR*, 2019.
[38] H. Qin, R. Gong, X. Liu *et al.*, "Forward and backward information retention for accurate binary neural networks," in *Proc. CVPR*, 2020, pp. 2247–2256.
[39] S. Jin, S. Di, X. Liang *et al.*, "Deepsz: A novel framework to compress deep neural networks by using error-bounded lossy compression," in *Proc. HPDC*, 2019, pp. 159–170.
[40] S. Guiaşu, "Weighted entropy," *Reports on Mathematical Physics*, vol. 2, no. 3, pp. 165–179, 1971.
[41] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5456–5464.
[42] X. Zhu, W. Zhou, and H. Li, "Adaptive layerwise quantization for deep neural network compression," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2018, pp. 1–6.
[43] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.
[44] F. Sattler, S. Wiedemann, K. Müller *et al.*, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *Proc. IJCNN*, 2019, pp. 1–8.
[45] S. U. Stich, J. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018, pp. 4452–4463.
[46] S. Gupta, T. Patel, C. Engelmann *et al.*, "Failures in large scale systems: long-term measurement, analysis, and implications," in *Proc. SC*, 2017, pp. 44:1–44:12.

[47] C. D. Martino, Z. T. Kalbarczyk, R. K. Iyer *et al.*, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *Proc. DSN*, 2014, pp. 610–621.
[48] K. Zhao, S. Di, S. Li *et al.*, "Ft-cnn: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2020.
[49] F. Zhang, J. Zhai, B. He *et al.*, "Understanding co-running behaviors on integrated cpu/gpu architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 905–918, 2016.
[50] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.
[51] A. Verma, L. Pedrosa, M. Korupoluet *et al.*, "Large-scale cluster management at google with borg," in *Proc. EuroSys*, 2015, pp. 18:1–18:17.
[52] S. Kavulya, J. Tan, R. Gandhi *et al.*, "An analysis of traces from a production mapreduce cluster," in *Proc. CCGRID*, 2010, pp. 94–103.
[53] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. NeurIPS*, 2019, pp. 8024–8035.
[54] K. He, X. Zhang, S. Ren *et al.*, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.
[55] M. Sandler, A. G. Howard, M. Zhu *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. CVPR*, 2018, pp. 4510–4520.
[56] X. Zhang, X. Zhou, M. Lin *et al.*, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR*, 2018, pp. 6848–6856.
[57] A. Krizhevsky and et al, "Learning multiple layers of features from tiny images," 2009.
[58] J. Jiang, F. Fu, T. Yang *et al.*, "Skcompress: compressing sparse and nonuniform gradient in distributed machine learning," *VLDB J.*, vol. 29, no. 5, pp. 945–972, 2020.
[59] H. Xu, K. Kostopoulou, A. Dutta *et al.*, "Deepreduce: A sparse-tensor communication framework for federated deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 150–21 163, 2021.

Haoyu Jin is currently working toward the MS degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include Federated Learning, Model Compression. He has published several papers in major journals and international conferences including the AAAI and ICCD.

Donglei Wu is currently working toward the PhD degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include Federated Learning, Model Compression. He has published several papers in major journals and international conferences including the AAAI and ICCD.

Shuyu Zhang received his master's degree in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include Federated Learning, Model Compression. He has published several papers in major journals and international conferences including the AAAI and ICCD.

Xiangyu Zou (Student Member, IEEE) is currently working toward the PhD degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include data deduplication, storage systems, lossy compression, etc. He has published 20+ papers in major journals and international conferences including IEEE-TPDS, ACM-TOS, FAST, USENIX ATC, ICDE, AAAI, etc.

Sian Jin (Student Member, IEEE) is currently working toward the PhD degree majoring in computer science at Washington State University. His research interests include High Performance Computing, Compression Algorithms, Artificial Neural Networks, and Parallel Computing. He has published several papers in major journals and international conferences including the PPoPP, HPDC, IPDPS and ICDE. etc.

Dingwen Tao is an associate professor at Indiana University, where he leads the High-Performance Data Analytics and Computing (HiPDAC) Lab. He is the recipient of various awards including NSF CAREER Award (2023), Amazon Research Award (2022), Meta Research Award (2022), RD100 Awards Winner (2021), IEEE Computer Society TCHPC Early Career Researchers Award for Excellence in HPC (2020), NSF CRII Award (2020), IEEE CLUSTER Best Paper Award (2018). He is serving on the Technical Review Board of IEEE Transactions on Parallel and Distributed Systems. He served as the Program Co-chair of 2021 IEEE International Conference on Scalable Computing and Communications and International Workshops on Big Data Reduction. He is also a reviewer, program committee member, or session chair of major HPC venues, such as SC, HPDC, ICS, IPDPS, CLUSTER, ICPP, CCGrid, HiPC, NPC. He has published in the top-tier HPC and big data conferences and journals, including SC, ICS, HPDC, PPoPP, DAC, PACT, IPDPS, CLUSTER, ICPP, BigData, IEEE TC, IEEE TPDS, etc.

Qing Liao (Member, IEEE) received the PhD degree from the Hong Kong University of Science and Technology, Hong Kong, in 2016. She is currently a Professor with the Department of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China, and also with the Department of New Networks, Peng Cheng Laboratory, Shenzhen. Her research interests include data mining, artificial intelligence, and information security.

Wen Xia (Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently an associate professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen. His research interests include data reduction, storage systems, cloud storage, etc. He has published more than 70 papers in major journals and conferences including the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, Proceedings of the IEEE, FAST, USENIX ATC, ICDE, AAAI, HotStorage, MSST, DCC, IPDPS, ICPP, ICCD, Cluster, etc.