BIRD+: Design of a Lightweight Communication Compressor for Resource-Constrained Distribution Learnin Pl f

Donglei Wu, Weihao Yang, Xiangyu Zou, Hao F

***Abstract*—The Top-K sparsification-based compression fra work is extensively explored for reducing communication cost distributed learning. However, we have identified several iss with existing Top-K sparsification-based compression meth severely impeding their applicability in resource-constrained p forms: (*i*) The limited compressibility of the Top-K paramet indexes critically restricts the overall communication compr sion ratio; (*ii*) Several time-consuming compression operati significantly offset the benefits of communication compressi (*iii*) The use of error feedback techniques to maintain mo quality results in a high memory footprint consumption.**

To solve these issues, we propose a lightweight tensor-w *Bi-Random sampling* strategy with an expectation invaria property called BIRD, which achieves higher communicat compression ratios at lower computational overheads wh maintaining a comparable model quality without incurring extra memory costs. Specifically, BIRD applies a tensor-wise *index sharing* mechanism that substantially reduces the proportion of the index by allowing multiple tensor elements to share a single index, thus improving the overall compression ratio. Additionally, BIRD replaces the time-consuming Top-K sorting with a faster *Bi-Random sampling* strategy at lower $O(N)$ time complexity based on the aforementioned *index sharing* mechanism, significantly reducing computational costs of compression; Moreover, BIRD establishes an *expectation invariance* property into the above *Bi-Random sampling* to ensure an approximate unbiased representation for the L_1-norm of the sampled tensors, effectively maintaining the model quality without incurring extra memory costs. We further optimize BIRD to BIRD+ by introducing the uniform distribution-based sampling and Gamma correction on the tensor-wise sampling process, achieving a more flexibly adjustment of the sparsity with better convergence performance.

Experimental evaluations across multiple conventional distributed learning tasks demonstrate that compared to state-of-the-art compression approaches, our proposed BIRD+ achieves higher communication compression ratios up to 36.2× and higher computation throughput up to 149.6× while maintaining the model quality without incurring extra memory costs.

***Index Terms*—Distributed learning, communication compression, random sampling, neural network.**

D. Wu, W. Yang, X. Zou, S. Li, W. Xia, B. Fang are with Harbin Institute of Technology, Shenzhen, Guangdong, China, 518055; Department of New Networks, Peng Cheng Laboratory, Shenzhen, China, 518055; Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen, China, 518055.

Hao Feng is with Indiana University, Bloomington, IN, USA, 47405.

Dingwen Tao is with State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, 100190.

E-mail: {donglei.wu, weihao.yang00}@hotmail.com, {zouxiangyu, lishiyi, xiawen}@hit.edu.cn, haofeng@iu.edu, taodingwen@ict.ac.cn, fangbx@cae.cn.

The preliminary manuscript has been accepted in ICCD'23. This journal version includes a more comprehensive design of our proposed approach. The main improvements include: (1) A novel L_∞-norm-based uniform random sampling that improves the model convergence stability and accuracy up to 0.88% than the BIRD in the conference version; (2) A controllable γ correction-based random sampling mechanism is developed to improve the communication compression ratio and throughput up to 10.75× and 3.35× than the BIRD in the conference version; (3) Experimental results and analysis on more conventional distributed learning datasets and models with various sparsity configurations.

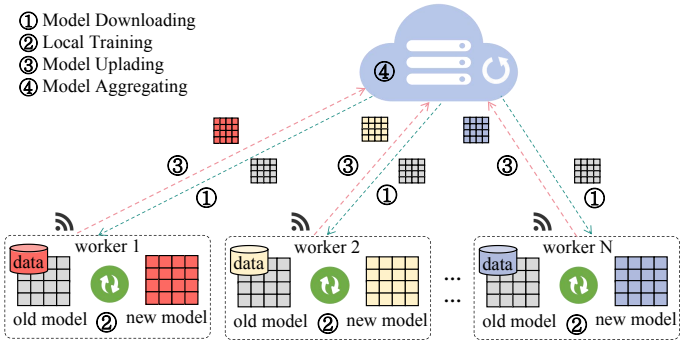


Fig. 1. The general workflow of Distributed Learning. The server in the cloud is served for receiving and aggregating the uploaded models, which are trained by each client. Numbers denote the order of each procedure in the training of Distributed Learning.

I. Introduction

With the rapid expansion of the size of Deep Neural Networks (DNN) and the training datasets in the field of Artificial Intelligence (AI), it has become inefficient to rely solely on centralized systems for training AI tasks [1], [2], [3]. Consequently, the deployment of DNNs in a decentralized environment has become imperative. Distributed learning with data parallelism has emerged as a prevalent approach to enhance performance and expedite DNN training by allowing multiple clients to collaboratively train a model on their local platforms or devices [4], [5], [6], [7], [8], [9].

In a typical distributed learning with data-parallelism scenario, such as Federated Learning (FL), training involves a multitude of Edge Devices, including smartphones and IoT devices, operating under constrained resources such as limited network bandwidth, computing power, and memory space [10]. Using the prevalent Parameter Server (PS) architecture as an example, Figure 1 illustrates the primary processes of distributed learning, broadly outlined in four steps: ① Clients download the current global model parameters (e.g., gradients or model updates) from the central parameter server. ② Clients independently train the downloaded global model

on their private datasets in parallel, generating a new loc model. ③ Clients upload their newly trained local models the server. ④ The server aggregates the received local mod parameters and generates a new global model, representi the training result of the current communication round. Aft the server sends back the averaged global model to clien they initiate the next training round by continuing to train tl received model with their local datasets. The above four ste are iteratively performed between clients and the server un the global model converges the training target of distribut learning.

In contrast to high-end in-cluster network infrastructur boasting abundant bandwidth, Edge Devices typically depe... on wireless connections characterized by constrained and fluctuating upload speeds, often substantially lower than download speeds [11], [12]. For instance, the average broadband speed in the United States was 55.0Mbps for downloads compared to 18.9Mbps for uploads. Notably, certain internet service providers exhibit even greater asymmetry, such as Xfinity with a speed of 125Mbps for downloads versus 15Mbps for uploads [13]. Meanwhile, the escalating performance of Deep Neural Networks (DNNs) has led to a rapid expansion in model size, outpacing the growth of available bandwidth. Consequently, the growing contradiction between expanding model dimensions and restricted bandwidth highlights the pressing need for communication compression techniques tailored to resource-constrained client devices in distributed learning.

To solve this low bandwidth challenge, Top-K Sparsification, a widely adopted compression technique is developed to reduce the communication costs from clients to the server in distributed learning [14]. Specifically, the client only selects and uploads a small part of 'important' parameters to the server based on the magnitude of the change before and after training. Typically, the selected elements are structured as a pair of $<$*the index*, *the value*$>$, where 'value' and 'index' maintain a 1:1 relationship in quantity. The *'index part'* denotes the coordinates of the transmitted parameters, which do not contribute to enhancing model performance. Conversely, the *'value part'* is utilized to update the parameters and plays a crucial role in enhancing the model.

For achieving a more substantial compression ratio, prior studies have employed lossy quantization for the *'value part'* and the lossless coding for the *'index part'* to further mitigate communication costs [10], [15], [16], [17]. Typically, the floating-point *'value part'* is quantized into lower-bit integers, while the *'index part'* is encoded into fewer bits. However, deploying existing Top-K sparsification-based compression approaches in large-scale distributed learning with numerous resource-constrained edge devices introduces three issues (as depicted in Figure 2): ❶ **Low Index Compression Ratio**: Current methods are limited by the compressibility of the *index part,'* which is often lower than the *value part'* and contributes more significantly to overall traffic [10], [18]. This limitation constrains the overall communication compression ratio. ❷ **High Compression Computational Cost**: Several computationally intensive compression processes, such as top-K sparsification sorting and lossless index coding, can incur substantial time costs, especially when computational

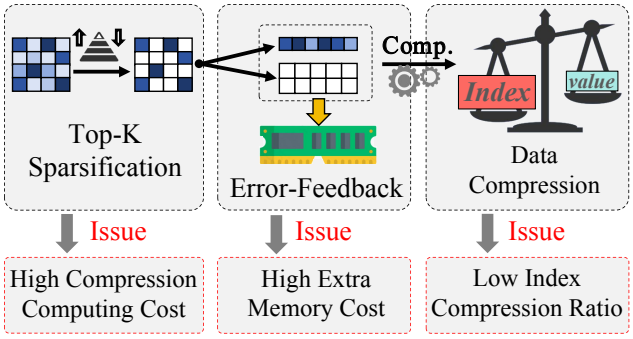


Fig. 2. Workflow and issues of traditional Top-K sparsification-based communication compression approaches.

resources like GPU kernels are limited [19]. These overheads diminish the benefits of communication compression. ❸ **High Extra Memory Cost**: Biased compression techniques, such as Top-K sparsification and 1-bit value quantization, often use error feedback mechanisms to maintain model quality. However, these techniques demand significant additional memory footprints to store dropped information, resulting in considerable extra memory costs [14], [15].

Hence, beyond pursuing a high compression ratio, a solid communication compression algorithm should also carefully consider the additional compression overhead and model quality, especially in the context of resource-constrained client devices. In this paper, we propose a lightweight communication compression technique BIRD, which tackles the aforementioned issues of performance and efficiency based on several observations and techniques, which are outlined below.

For the **Low Index Compression Ratio issue**, we observe that multiple elements within a DNN's tensor maintain a fixed relative position relationship. Therefore, one tensor index is enough to determine all element's positions within this tensor if this tensor is uploaded to the server. Exploiting this characteristic, we propose a tensor-wise *index sharing* mechanism, allowing a single tensor index to be shared among all elements within this tensor by changing the granularity from element to tensor. This innovation effectively reduces the index-to-value proportion from 1:1 to 1:n, resulting in a substantial enhancement of the overall communication compression ratio.

For the **High Compression computational cost issue**, we mitigate the time cost of the sparsification process and I/O by developing a lightweight *Bi-Random sampling* strategy, which replaces the time-consuming element-wise Top-K sorting with a faster tensor-wise random[1] sampling algorithm [22]. Coupled with the previously mentioned tensor-wise *index sharing* mechanism, the object of *Bi-Random sampling* strategy becomes tensor, resulting in very few index data left, thus further diminishing the compression overhead of the index.

For the **High Extra Memory Cost issue**, we draw inspiration from existing stochastic sparsification techniques that uphold model quality without requiring error feedback, preserving the unbiased statistical property of the original [23], [20]. Motivated by this feature, we establish an approximate

[1]Notice that the 'random' doesn't mean exactly equal probabilities, but can be guided by the probabilities based on the parameter's value[20], [14], [21]

expectation invariance property in the aforementioned *Bi-Random sampling*. By maintaining statistical unbiasedness for the L_1-norm of the original tensor, we can achieve a comparable training quality of distributed learning without incurring extra memory costs.

Although the above techniques can effectively reduce communication costs at a low computational overhead, BIRD still has two limitations: (1) the approximate *expectation invariance* property leads to an unstable model convergence (will be introduced in detail in Section V-F). (2) The fully adaptive sampling algorithm potentially leads to a high communication cost while achieving an unpredictable low sparsity. To address these limitations, we further optimize BIRD to BIRD+ by introducing a uniform distribution-based sampling algorithm with sparsity controllable probability Gamma Correlation strategy. This approach not only achieves a more stable model convergence performance under a strict expectation invariance property, but also obtains a higher communication compression ratio by tuning an appropriate sparsity.

In summary, we propose a lightweight distributed learning compressor BIRD+, by developing a **BI**-**R**an**D**om sampling strategy with an expectation invariance property based on a tensor-wise index sharing mechanism, to address the Low Index Compression Ratio issue, High Compression Computational Cost issue, and High Extra Memory Cost issue. BIRD+ achieves high compression ratios with low computational costs while maintaining model quality without incurring extra memory costs. Our contributions can be summarized as follows:

- We identify three critical issues in the Top-K sparsification-based compression workflow: (*i*) the overall compression ratio of existing methods is limited by the low compressibility of the learning-useless index part; (*ii*) computationally expensive operations, such as sorting and index coding, compromising the advantages of communication compression; (*iii*) the error feedback mechanism, essential for maintaining the model training quality, incurs significant extra memory costs equivalent to the size of the full model.
- To tackle these issues, we propose a novel distributed learning compressor, named BIRD+, tailored for resource-constrained edge devices by: (*i*) Introducing a tensor-wise *index sharing* mechanism to significantly decrease the index proportion, thereby improving the overall compression ratio; (*ii*) Developing a lightweight *Bi-Random sampling* strategy based on the *index sharing* mechanism, effectively reducing the computational costs of compression; (*iii*) Establishing a tensor-wise *expectation invariance* property within the above *Bi-Random sampling* strategy for the sampled tensors, maintaining the model quality without incurring extra memory cost.
- We evaluated the performances of BIRD+ and conducted comparisons with different state-of-the-art distributed learning compression approaches across multiple mainstream tasks for three aspects: compression ratio, computational efficiency, and model quality. Experimental results demonstrate that (*i*) BIRD+ achieves a higher overall communication compression ratio up to 32.6×; (*ii*) significantly reducing time complexity from $O(NlogK)$ to $O(N)$, resulting in a higher computation throughput up to 149.6×; (*iii*) effectively maintaining model quality without incurring extra memory costs.

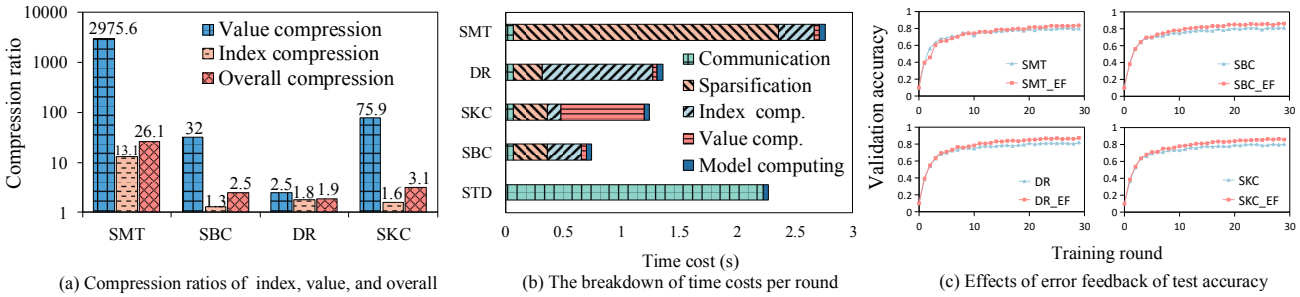
II. Background and Related Works

In the era of Artificial Intelligence and Big Data, the exponential growth of datasets and model parameters has promoted the development of distributed learning frameworks for training modern Deep Neural Networks (DNNs), which distribute computational tasks across various devices to enhance training efficiency. However, challenges arise due to the substantial number of clients, restricted uplink network bandwidth in client devices, and the expanding model size, causing the communication problem as a bottleneck in the practical resource-constrained distributed learning environment, such as smartphones and IoT devices.

To reduce the communication cost of distributed learning, the Federated Averaging (FedAvg) [24] algorithm emerges as a solution, which lowers the communication frequency by shifting bandwidth-intensive processes to local computation. Building upon FedAvg, additional techniques such as sparsification, quantization, and encoding are exploited to further compress the size of transmitted data (e.g., the model weight updates or the gradients) between the parameter server and the clients.

Quantization-based approaches: These strategies alleviate communication overhead by encoding uploaded parameters into fewer bits. Typically, *Seide et al.* [25] introduce the 1-bit quantization technique, where each 32-bit floating-point parameter in a neural network is condensed into 1-bit, capturing its sign. It is then decoded using the sign and mean value of the neural network's parameters. To maintain the model accuracy and convergence rate, quantization errors accumulate and are compensated to the encoding in the next round. *Wen et al.* [23] proposed a statistic-based stochastic algorithm, called TernGrad, to encode the transmitted floating-point gradients with two bits, while a theoretically-guaranteed convergence is proven. Alistarh *et al.* [21] extended the statistic-based quantization method by smoothly balancing the trade-off between communication bandwidth and convergence. Although quantization-based approaches achieve a maximum compression ratio of around 32×, sparsification-based communication compression techniques are proposed to further enhance compression ratios.

Sparsification-based approaches: Top-K sparsification is a prevalent technique in mitigating communication overhead in distributed learning by transmitting only a fraction of the gradients or parameter updates with the largest magnitude to the server [17], [16]. To ensure model quality, the unselected parameters, termed residuals, accumulate locally and are reintroduced in the subsequent round. Specifically, the residual model has the same size as the full model (e.g., about 100MB for Resnet50[26]), will be stored in the memory and added back into the next round in an element-wise manner. The selected elements are structured as $<$*the index*, *the value*$>$, which can be compressed through quantization and encoding techniques to further reduce communication costs. For in-



(a) Compression ratios of index, value, and overall (b) The breakdown of time costs per round (c) Effects of error feedback of test accuracy

Fig. 3. The key observations with experimental validation on different Top-K sparsification-based the-state-of-the-art compressors.

stance, DGC [16] utilizes Top-K sparsification to reduce gradient size and employs run-length coding to compress the traffic of sparse gradient indices. Additionally, DGC incorporates momentum correction, local gradient clipping, momentum factor masking, and warm-up training to address accuracy loss and staleness problems caused by reduced communication. Moreover, *Chen et al.* [27] present the Adaptive Residual Gradient Compression scheme, dynamically tuning the compression ratio by deciding the maintained length of residual gradients through sparsification. However, we notice that the communication cost of the index, the computational cost of some compression processes, and the memory cost of residuals impede the deployment of these Top-K sparsification-based compression methods in typical distributed learning scenarios involving numerous resource-constrained devices.

Recent developments in distributed learning introduce stochastic sparsification-based communication compression algorithms to reduce gradient communication costs [20], [14], [28]. These algorithms establish a positive relationship between the probability of an element being randomly sampled and its magnitude. In these approaches, a scaling factor is applied to the sampled components to align the expectation of the compressed model with the raw model. Unselected elements are dropped by setting them to zero. Crucially, as unbiased compression algorithms, these methods require no additional error feedback to maintain model convergence, thereby avoiding extra memory overhead. Leveraging this unbiased property, we propose a tensor-wise random sampling algorithm characterized by an expectation invariance property, achieving a lightweight communication compression without incurring additional memory costs.

Hybrid approaches: Quantization and sparsification, being two orthogonal methods, are often synergistically combined with a lossless encoding scheme to comprehensively compress communication costs in distributed learning. For example, *Sattler et al.* [10] introduced Sparse Binary Compression (SBC), a communication-efficient distributed learning framework. SBC combines Top-k sparsification, quantization, and encoding. It employs Top-k sparsification by setting all weight updates, except the top k with the highest magnitude, to zero. The sparse weight updates are then quantized to binary, and the index of non-zero elements is encoded using optimal Golomb encoding[29], achieving an additional $1.9\times$ compression ratio. *Strom et al.* [30] combined sparsification and 1-bit quantization techniques to compress Fully Connected (FC) layers. Golomb coding is also employed on the index part to further enhance compression. Similarly, *Dryden et al.* [31] fused sparsification and quantization, introducing a percentage threshold in sparsification and using the mean value of the selected parameters as the quantized result. *Tsuzuku et al.* [32] proposed a variance-based sparsification algorithm to compress gradients in distributed learning. Additionally, this approach further applies 4-bit quantization and Golomb encoding to improve the overall communication compression ratio. Recently, SKC [18] utilized the sketching technique with bucket quantization to obtain a high value compression ratio and applied the prefix technique to compress the index. DR [33] employs a novel value-fitting algorithm to reduce value costs, and a bloom filter is used to compress the index.

In summary, prevalent communication compression approaches in distributed learning often integrate three orthogonal techniques: sparsification, quantization, and encoding. Sparsification is used to reduce the number of uploaded elements, quantization is used to further compress the cost of the 'value' part, and lossless encoding is used to minimize the cost of the 'index' part. In this paper, we propose multiple lightweight techniques to achieve a considerable communication compression ratio more efficiently.

III. Key Observations and Motivations

In this section, we elucidate our key observations and motivations through the following approaches: (1) Training VGG16 [34] on CIFAR10 [35] in the distributed learning context with 10 clients and 1 server[2]. (2) Implementing the compression of locally trained model updates by utilizing four state-of-the-art Top-K sparsification-based compressors.

(i) In traditional Top-K sparsification, the selected parameters within the DNN model are structured as $<$*the value*, *the index*$>$ pair with the **1:1** quantitative relationship. We notice that ① the overall compression ratio per parameter (cr_z) is constrained by either the compression ratio of the index (cr_i) or the value (cr_v). Assuming a fixed

[2]The communication module is built on the collective communication of torch.distributed v.1.10.2 package, utilizing the Gloo backend under the 1 Gbps network bandwidth.

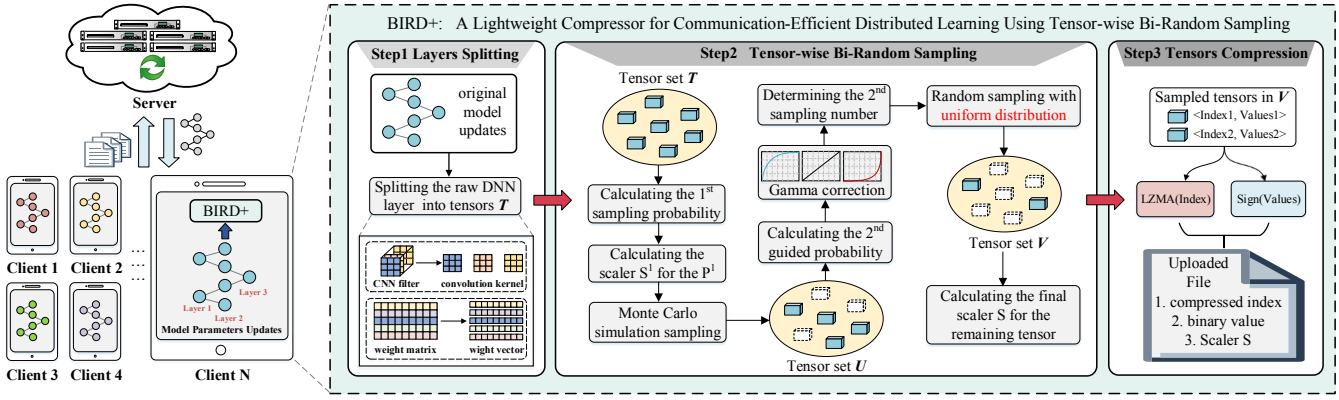


Fig. 4. The workflow of the distributed learning system with our proposed communication compression algorithm, BIRD+, involves three steps: (1) Layer Splitting, (2) Bi-Random Sampling, and (3) Tensors Compression.

compression ratio of α on both sides, the limitation on the overall compression ratio can be derived as follows:

$$\lim_{cr_v \to \infty} cr_z = \frac{1}{\frac{0.5}{cr_i} + \frac{0.5}{cr_v}} = 2\alpha \qquad with \qquad cr_i = \alpha \quad (1)$$

Eq. 1 states that restricting either index or value compression will confine the overall compression ratio, adhering to *Cannikin's law*. ② The index part accounts for a considerable portion of communication costs in most sparsification-based compression methods [16], [10], [18]. As depicted in Figure 3 (a), while the value can be compressed to a *low-precision lossy version* with a high compression ratio of 16×-32×, the index undergoes only lossless compression with a restricted compression ratio of 2×-3× to ensure precise matching of the value. Consequently, the overall compression ratio is generally limited to no more than twice that of the index, aligning with Eq. 1. Evidently, allocating a substantial portion of communication costs to transmit learning-useless coordinate information (i.e., the index part) is impractical.

The aforementioned findings prompt us to reduce the **1:1** proportion relationship between the index and value to a more advantageous **1:n** by regarding the tensor as a fundamental sparse object for *each layer*, as opposed to a traditional individual element of a flattened DNN vector (element-wise → tensor-wise). In detail, we initially partition each layer into tensors following specific rules, with each tensor containing multiple parameters. Subsequently, we identify several 'important' tensors for further compression and upload. In this approach, **one index is shared across all parameters in a tensor**, leading to a significant reduction in the communication costs associated with the index part and an enhancement in the overall compression ratio.

(ii) The primary objective of communication compression is to accelerate the distributed learning process. However, due to the inherent limitations in computing power on edge devices (such as insufficient GPU kernels), certain compression operations incur significant time costs [19], [22]. Figure 3 (b) illustrates the average time breakdown per training *iteration* during distributed training. Experimental results indicate that while model compression results in reduced communication time in distributed learning, the runtime of several resource-intensive compression processes (e.g., sparsification and index compression) can counterbalance the benefits derived from the decreased communication time. In some instances, communication compression may even extend the overall duration of the standard distributed training (STD) process, implying a negative communication compression gain.

This observation impels us to substitute the conventional yet costly element-wise Top-K sparsification with an innovative tensor-wise random sampling approach. To be specific, the parameters selection process (i.e., sparsification) is performed by randomly sampling tensors, with the sampling probability of each tensor being proportional to its magnitude. In doing so, the time complexity of sparsification is reduced from typical $O(Nlogk)$ to $O(N)$ [22]. Moreover, since the object of our random sampling strategy is tensor, both the overheads of data access and the subsequent index compression are diminished.

(iii) Top-K sparsification is a *biased* lossy compression technique. To maintain the accuracy of the compressed model in Top-K sparsification, an *Error Feedback (EF)* mechanism [14] is introduced to locally accumulate the unselected elements (referred to as residuals) and add them back in the subsequent round. Figure 3 (c) illustrates that the different state-of-the-art Top-K methods exhibit varying degrees of test accuracy degradation (3.28%-5.55%) without *EF*, underscoring the crucial role of EF in maintaining model quality. Nevertheless, the Error Feedback mechanism necessitates additional memory space equivalent to the full model (∼61MB), posing constraints on scalability for edge devices with limited memory resources, such as smartphones and IoT devices.

Motivated by unbiased stochastic sparsification techniques [23], [20], which achieve considerable convergence rates without the need for *EF* by maintaining the expectation invariance of sampled elements, we establish an expectation invariance property for the sampled tensors in the aforementioned tensor-wise random sampling. Specifically, we preserve the unbiasedness of the tensor's L_1-norm by **calculating a scaler $\mathcal{S}$ based on the sampling probability, which is used to represent the magnitude of the sampled tensors**

during the recovery stage on the server. Consequently, the model quality is maintained without relying on the *EF* to store the extra residuals in the client's memory.

IV. Design and Implementation

A. Overview

In this paper, we propose BIRD+, a lightweight communication compression algorithm designed to reduce the communication cost of clients in distributed learning. Figure 4 illustrates the three key steps involved in BIRD+:

1) **Layer Splitting** (1) Splitting each layer of the model into multiple tensors. (2) Calculating the L_1-norm and L_∞-norm for each tensor, which are used to guide BIRD+ to randomly sample the uploaded tensors.
2) **Tensor-wise Bi-Random Sampling**. (1) Conducting the first randomly sampling tensors for each layer based on their L_1-norm. (2) Conducting a second random sampling on the remaining tensor under a uniform distribution guided by their L_∞-norm. (3) Calculating a scaler $\mathcal{S}$ for each layer, which is used to recover an unbiased representation of the original full-precise model on the server side.
3) **Tensors Compression**. (1) Quantizing the values of the elements in the sampled tensors to binary based on the signs. (2) Encoding the index of the sampled tensors with fewer bits using the LZMA lossless compressor.

Subsequent subsections will delve into a comprehensive introduction of each step in BIRD+.

B. Layer Splitting

Motivated by the 1^{st} and the 2^{nd} key observations, we introduce a tensor-wise *index sharing* mechanism, treating the tensor as a fundamental sparsification object to achieve two primary objectives: (*i*) reducing the index costs by sharing an index among all elements of a tensor and (*ii*) lowering the computational costs of data I/O and subsequent index compression. Given that in existing DNNs, (1) parameters between neighboring layers are commonly organized in the shape of #Input $\times$ #Output, and (2) parameters in the front layer are often employed as a monolith for feature extraction, we propose to split the tensor of each DNNs layer based on the following rules:

- **4-D layer**: For the typical filter in the convolutional layer with four dimensions $< N, W, H, C >$, we consider a 2-D convolution kernel with $W \times H$ elements as a fundamental operated tensor, resulting in a total of $C \times N$ tensors in this convolutional layer. In this scenario, $W \times H$ elements within one tensor share one index.
- **2-D layer**: For the typical weight matrix with two dimensions $< I, O >$ of a fully connected matrix or *Q, K, V* matrices of a transformer structure, each column of a matrix with I elements is considered as a fundamental operated tensor, yielding a total of O tensors. Similarly, these I elements share one index within this tensor.
- **1-D layer**: For the typical bias or BN layer, each element is associated with a specific 2-D convolution kernel or weights matrix's vector from the front layer. Therefore, 1-D Tensors are not split and will be selected if the corresponding attached tensor is sampled.

By performing layer-by-layer splitting of the DNN model, the elements of the DNN tensor are grouped into 'chunks' for subsequent random sampling and compression.

C. Element-Wise Random Sampling

The 2^{nd} and the 3^{rd} observations indicate that the Top-K sparsification-based compression techniques with error feedback not only result in non-negligible computational costs but also incur extra memory costs for the dropped parameters (i.e., residual accumulation). To address these issues, we propose a more efficient Bi-Random sampling strategy with an expectation invariance property based on the above tensor-wise index sharing mechanism to replace the traditional Top-K sparsification technique. To facilitate a better understanding of our methodology, let's first take a close look at the Element-Wise random sampling process.

Element-wise random sampling-based sparsification techniques have been studied for achieving low-precision processing with a provable guarantee of convergence [23], [20], [21]. Formally, given a floating point vector $\boldsymbol{g} \in \mathbb{R}^d$, the i-th component g_i is calculated as:

$$\widetilde{g_i} = \frac{g_i}{p_i} \times b_i \quad with \quad \begin{cases} P(b_i = 1|g_i) = p_i \\ P(b_i = 0|g_i) = 1 - p_i \end{cases} \tag{2}$$

where the binary vector $\mathbf{b} \in \{0, 1\}^d$ indicates whether the i-th element will be sampled: $b_i = 1$ with probability p_i and $b_i = 0$ with probability $1 - p_i$. The sampling probability is defined as $p_i = \frac{|g_i|}{max(|\mathbf{g}|)}$. Practically, the sampling process could be efficiently achieved using the insight of Monte Carlo simulation. As illustrated in Figure 5, the value of b_i could be determined by comparing the p_i with a pseudo-random floating-point number of the uniform distribution. Intuitively, **the larger the magnitude of the parameter, the more likely it is to be selected**. As a result, the sampled elements (i.e., $b_i \neq 0$) are formed as:

$$\widetilde{g_i} \setminus \mathbf{0} = \frac{g_i}{p_i} = g_i \times \frac{max(|\boldsymbol{g}|)}{|g_i|} = max(|\boldsymbol{g}|) \times sign(g_i) \tag{3}$$

Eq. 3 suggests that elements of the raw vector $\boldsymbol{g}$ are randomly transformed to 0 or $\pm max(|\mathbf{g}|)$ with *unbiased estimation* as:

$$\mathbb{E}[\widetilde{g_i}] = p_i \times \frac{g_i}{p_i} \times 1 + (1 - p_i) \times 0 = g_i \tag{4}$$

The theory proves that unbiased random sparsification can efficiently compress the DNNs vector without significant degradation in model accuracy [23], [20], [21]. In contrast to the Top-K sparsification-based method, random sampling has a lower $O(N)$ time complexity without incurring extra memory costs. Leveraging this characteristic, we propose a Bi-Random sampling strategy with an expectation invariance property based on the tensor-wise index sharing mechanism, as detailed in the next subsection.

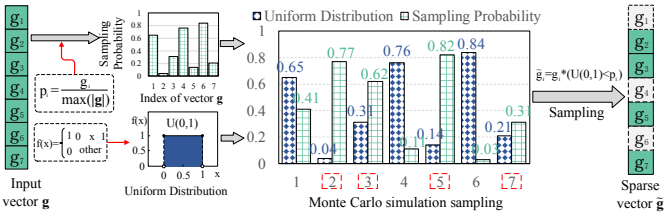


Fig. 5. A general example of Monte Carlo simulation sampling process.

D. Tensor-Wise Bi-Random Sampling

As introduced in sections I and II, the unbiased stochastic sparsification techniques derive a sparse model by assigning higher probabilities to elements with larger magnitudes during the sampling process. To maintain model training quality without accumulating the un-sampled elements (i.e., residual model), the sampled elements are multiplied by a scalar $\mathcal{S}$, ensuring the unbiased expectation of the model. Inspired by this feature, we introduce a Bi-Random sampling strategy.

1) L_1-norm based random sampling: After splitting the model parameters into tensors, the magnitude of each tensor t_i can be measured using its L_1-norm, and tensors can be randomly sampled based on their L_1-norm layer-by-layer. To maintain the model's convergence rate, we establish an *expectation invariance* property by ensuring that the expectation of the L_1-norm for the sampled layer remains unchanged.

Specifically, assuming that a full precise floating-point layer is divided into K tensors: $\boldsymbol{T} = \{t_1, \cdots, t_K\}$, the probability of selecting the i-th tensor t_i is given by $p_i^1 = \frac{||t_i||_1}{max(||\boldsymbol{T}||_1)}$, where $||\cdot||_1$ returns the L_1-norm of the tensor. To maintain the model convergence rate, we aim to preserve the unchanged expectation for the L_1-norm of the layer as follows:

$$t_i^1 = \underbrace{sign(t_i) \times b_i}_{sampled\ t_i} \times \underbrace{\frac{max(||\boldsymbol{T}||_1)}{m}}_{scaler\ \mathcal{S}^1} \tag{5}$$

where $sign(\cdot)$ returns all signs of all elements in a tensor (e.g., from $< -0.05, 0.24, -0.38 >$ to $< -1, 1, -1 >$), and the binary value $b_i \in \{0, 1\}$ indicates whether tensor t_i will be sampled. In practice, the element value of b_i could be efficiently obtained by Monte Carlo simulation. The term $\mathcal{S}^1 = \frac{max(||\boldsymbol{T}||_1)}{m}$ represents the magnitude of the elements in the non-zero tensor being sampled, which is used to recover the trainable floating-point tensors in the server. t_i^1 denotes the i-th tensor with m elements (e.g., a sampled 3×3 convolution kernel has $m = 9$). Eq. 5 ensures a full precise floating-point tensor t_i of a split layer will be randomly quantized to zero or its signs while preserving the expectation of the L_1-norm unchanged for t_i (known as the expectation invariance property) because:

$$\begin{aligned}
\mathbb{E}||t_i^1||_1 &= ||sign(t_i) \times \frac{max(||\boldsymbol{T}||_1)}{m} \times 1||_1 \times p_i^1 + (1 - p_i^1) \times 0 \\
&= ||sign(t_i)||_1 \times \frac{max(||\boldsymbol{T}||_1)}{m} \times \frac{||t_i||_1}{max(||(\boldsymbol{T})||_1)} \\
&= m \times \frac{max(||\boldsymbol{T}||_1)}{m} \times \frac{||t_i||_1}{max(||(\boldsymbol{T})||_1)} = ||t_i||_1
\end{aligned} \tag{6}$$

To facilitate the understanding of the above sampling process, a specific use case of the above sampling process is provided. Assuming that we have obtained five 1-dimension vectors [1,-1], [-2,2], [3,-3], [-4,4], [5,5] in a 2-dimension layer after the layer splitting. Our sampling algorithm first calculates their L_1-norm by summing the absolute value of each element. The computing results are [2, 4, 6, 8, 10]. Then we compute a probability vector with elements of [0,1] by dividing all L_1-norm values by the maximum. Thus the resulting vector is [2/10, 4/10, 6/10, 8/10, 10/10]. Each element of this probability vector represents the sampling probability of the corresponding vector, and the sampled vector can be efficiently obtained by Monte Carlo simulation.

In summary, the aforementioned L_1-norm based random sampling aims to randomly sample tensors with larger L_1-norm values at a higher probability while preserving an unchanged expectation of the L_1-norm (i.e., Eq. 5 and Eq. 6). This design allows us to achieve model sparsification at a low computational cost while maintaining the model training quality without incurring extra memory costs for the residuals.

2) Uniform distribution-based random sampling: According to the aforementioned L_1-norm based random sampling described in Eq. 5, the set of sampled non-zero tensors (denoted by $\boldsymbol{U} = \left\{t_1^1, \cdots, t_{|U|}^1\right\}$) is likely to have a large L_1-norm. **However, we observe that the sampling ratio generated by the aforementioned adaptive L_1-norm based random sampling process is not adequately low, indicating limitations in the achieved overall communication compression ratios**. To address this problem, we propose a second sampling stage to sample the remaining non-zero tensor set $\boldsymbol{U}$ again for further reducing the tensor sampling ratio and improving the overall communication compression ratio.

Motivated by the conventional Top-K sparsification algorithm [18], [10], [16], which prioritizes elements with the largest magnitude, a naive solution is to utilize the L_∞-norm, reflecting the maximum magnitude of a tensor, to measure the importance of the tensors in $\boldsymbol{U}$. Consequently, the remaining tensors in $\boldsymbol{U}$ with larger L_∞-norm are more likely to be selected. Similar to the first L_1-norm based sampling, this 'guided probability' of the j-th tensor t_j^1 in $\boldsymbol{U}$ is defined as $p_j^\infty = \frac{||t_j^1||_\infty}{max(||\boldsymbol{U}||_\infty)}$. By using the Monte Carlo simulation again, we can obtain the K sampled tensors (as BIRD do).

Obviously, the second L_∞-norm based sampling introduces changes to the expectation of L_1-norm of $\boldsymbol{V}$. However, since the elements' value of the tensor in $\boldsymbol{U}$ have been quantized to the signs (i.e., ± 1) after the first stage's L_1-norm based sampling, we cannot utilize the same strategy (i.e., Eq. 5) used to calculate the scalar $\mathcal{S}^1$ in this stage to preserve the expectation invariance property. To approximately ensure the expectation of the L_1-norm of $\boldsymbol{V}$ remains consistent with $\boldsymbol{U}$, an naive solution is to scale the magnitudes of sampled tensors again by multiplying $\mathcal{S}^1$ with a factor $\frac{|\boldsymbol{U}|}{|\boldsymbol{V}|}$, where $|\cdot|$ denotes the number of tensors. In doing so, the resulting scaler is defined as $\mathcal{S} = \mathcal{S}^1 \times \frac{|\boldsymbol{U}|}{|\boldsymbol{V}|}$.

Although the communication compression ratio can be improved by further sampling these K tensors, there are two limitations in the above second sampling process: (1) the strict

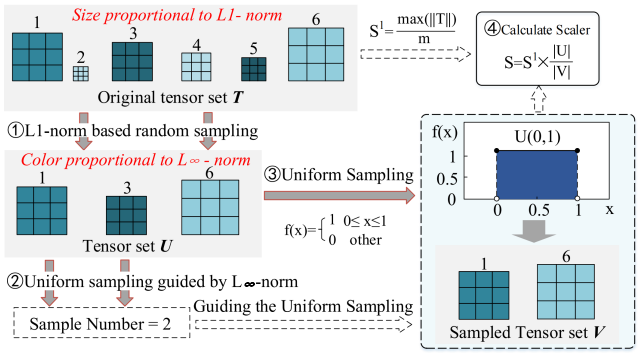


Fig. 6. A general example of Bi-Random Sampling process in BIRD+.

expectation invariance cannot be built on these tensors sampled under the L_∞-norm, this leads to the convergence performance might be degraded by elements with very large magnitude. (2) The fully adaptive sampling process potentially leads to a high communication cost as appearing an unpredictable low sparsity. In the following, we propose a uniform distribution-based sampling algorithm with controllable sparsity in the second sampling stage to address these limitations.

To improve the convergence stability of our sampling algorithm, we do not directly consider these K sampled tensors as the final sampled tensor, but set K as the number of sampled tensors under a uniform distribution-based sampling process. In doing so, K tensors will be sampled from $\boldsymbol{U}$ with an equal sampling probability. More importantly, we can improve the convergence stability by building a strict expectation invariance property in this uniform distribution-based sampling using the above scale method. Specifically, to ensure the expectation of the L_1-norm of $\boldsymbol{V}$ remains consistent with $\boldsymbol{U}$, the magnitudes of sampled tensors will be amplified by multiplying $\mathcal{S}^1$ with a factor $\frac{|\boldsymbol{U}|}{|\boldsymbol{V}|}$, where $|\cdot|$ denotes the number of tensors. Thus the final scaler is defined as $\mathcal{S} = \mathcal{S}^1 \times \frac{|\boldsymbol{U}|}{|\boldsymbol{V}|}$. Such that the expectation of tensor's L_1-norm in the scaled $\boldsymbol{V}$ can be derived as:

$$\mathbb{E}(|| \underbrace{t_j^1 \times \frac{|\boldsymbol{V}|}{|\boldsymbol{U}|}}_{\mathbb{E}||t_j^\infty||_1} ||_1 \times \frac{|\boldsymbol{U}|}{|\boldsymbol{V}|}) = \mathbb{E}||t_j^1||_1 \tag{7}$$

Eq. 7 suggests that the L_1-norm of the final sampled and scaled tensors remain unchanged. Different from traditional Top-K sparsification, our proposed Bi-Random sampling strategy, involving two random sampling stages (as illustrated in Figure 6), is a lightweight and unbiased communication compression algorithm.

To more flexibly control the sparsity, we introduce a *Gamma Correlation* [36] on the p_j^∞ to achieve various compression levels. Specifically, the sampling probability of the second L_∞-based random sampling is calculated as $p_j^\gamma = (p_j^\infty)^\gamma$. The hyper-parameter γ is used to control the sparsity. Note that the definition of sparsity is $\frac{d-s}{d}$, where s and d are the number of sampled parameters and parameters in the entire model. Higher sparsity (i.e., increasing compression ratio) can be achieved by setting $\gamma < 1$, and lower sparsity (i.e., decreasing compression ratio) can be achieved by setting $\gamma > 1$. Consequently, Gamma Correction enables a more flexible and controllable sparsity and communication compression. Note that Gamma Correction applied in this second sampling stage does not compromise the expectation invariance property as $|\boldsymbol{U}|$ and $|\boldsymbol{V}|$ will also change accordingly.

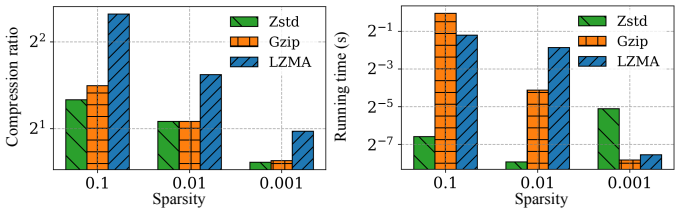
Algorithm 1: The tensor-wise Bi-Random sampling.

Input: The set of full precise floating-point tensors $\mathbf{T}$ of each DNN layer.
Output: The sampled tensors $\mathbf{V}$ and scaler S of each layer.
begin
  for $t_i \in \boldsymbol{T}$, i = 0, 1, 2, . . . **do**
    $p_i^1 \longleftarrow \frac{||t_i||_1}{max(||\mathbf{T}||_1)}$;
  $\mathbf{P}^1 \longleftarrow \{p_0^1, p_1^1, ..., p_{|T|}^1\}$;
  $\mathbf{U} \longleftarrow \mathbf{T} \odot (rand(0,1) < \mathbf{P}^1) \setminus \mathbf{0}$;
  $\mathrm{S}^1 = \frac{max(||\mathbf{T}||_1)}{m}$;
  for $t_j^1 \in \boldsymbol{U}$, j = 0, 1, 2, . . . **do**
    $p_j^\infty \longleftarrow \frac{||t_j^1||_\infty}{max(||\mathbf{U}||_\infty)}$;
    $p_j^\gamma \longleftarrow GammaCorrection(p_j^\infty)$;
  $\mathbf{P}^\gamma \longleftarrow \{p_0^\gamma, p_1^\gamma, ..., p_{|U|}^\gamma\}$;
  K $\longleftarrow |\mathbf{U} \odot (rand(0,1) < \mathbf{P}^\gamma) \setminus \mathbf{0}|$;
  Index $\longleftarrow$ *UniformSampling*(0, K-1);
  $\boldsymbol{V} \longleftarrow \mathbf{U}[Index]$;
  S = $\mathrm{S}^1 \times \frac{|\mathbf{U}|}{|\mathbf{V}|}$;
  return $\boldsymbol{V}, \boldsymbol{S}$

To facilitate the understanding of the above sampling process, we continue to discuss a specific use case based on the earlier case. Specifically, assuming that we have obtained two 1-dimension vectors [3,-3], [-5,5] after the above L_1-norm based random sampling. Our algorithm samples again from the remaining 3 vectors [2,-1], [-2,2], [-4,4]. To obtain the sampling numbers, BIRD+ first perform the L_∞-norm-based random sampling, which has the almost same steps as the L_1-norm based random sampling, except for replacing the computation of the L_1-norm with L_∞-norm. Assuming that we obtain two vectors from this L_∞-norm-based random sampling, we don't directly select these two vectors as the final sampled vector but randomly sample two vectors from [2,-1], [-2,2], [-4,4] under the uniform distribution.

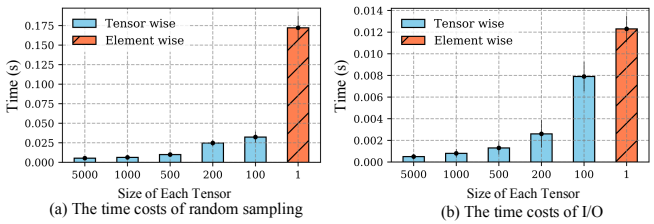
In summary, the aforementioned L_∞-norm-based random sampling is proposed to sample tensors again from set $\boldsymbol{U}$ because we notice that the L_1-norm based random sampling retains too many tensors, resulting in a low compression ratio. However, through L_1-norm based sampling, the first indicator L_1-norm is no longer applicable since the remaining tensor's L_1-norm has been quantized to the same value (i.e., ± 1) by Eq. 5. To achieve a more flexible and effective sampling from the remaining tensors, we first adaptively obtain the sampling number K from the remaining tensors guided by their L_∞-norm. Then, we randomly sample K tensors from the remaining tensors under the uniform distribution with the magnitude scaling method. Finally, we leverage a Gamma Correction strategy on the second sampling stage to achieve the controllable sparsity and compression ratio.

It is worth noting that our proposed tensor-wise sampling



(a) Evaluations of index compression ratio (b) Evaluations of index compression time

Fig. 7. Evaluations of compression ratio and runtime(s) of different lossless compressors under the various sparsity levels.



(a) The time costs of random sampling (b) The time costs of I/O

Fig. 8. The time costs of (a) sampling operation and (b) I/O under the different tensor's size.

approach reduces the communication costs of index data at the costs of increasing the risk of sampling some small parameters while sampling more large parameters obtains better training performance, as mentioned in many element-wise Top-K sparsification approach. To maintain the model training quality, our proposed Bird+ minimizes the negative effects of sampling small parameters by (1) Sampling the tensor with large L_1-norm at a high probability, suggesting that elements of the sampled tensors have large absolute values at a high probability. (2) Building a tensor-wise expectation invariance property within the sampling algorithm, and the expectation invariance property has been proven to be effective in maintaining model training quality. The detailed Bi-Random sampling processes of BIRD+ are summarized in the pseudo-code of Algorithm 1.

E. Tensors Compression

1) Value quantization: According to Eq. 5, the remaining tensors have been quantized to their signs layer-by-layer (i.e., 1-bit per element) through the aforementioned Bi-Random sampling processes. For each layer, a scaler $\mathcal{S}$ is calculated to represent the magnitude of the quantized tensors, which is used to recover the floating-point model in the server.

2) Index coding: Generally, the integer indices of the sampled tensors are highly redundant, thus we can further reduce their size by using a lossless compression technique. To achieve efficient compression for the index part, we evaluated the compression ratio and runtime of three typical lossless compression schemes: Zstd, Gzip, and LZMA, under various sparsity. Experimental results in Figure 7 suggest that *(i)* LZMA has a significant advantage in compression ratio over Zstd and Gzip. *(ii)* The compression times of LZMA are negligible when the volume of compressed data is not significant (i.e., under a low sparsity). Considering that the index sharing mechanism of BIRD+ can greatly reduce the number of indexes, indicating that the compressed data is limited, thus *LZMA* is the appropriate lossless compressor for BIRD+ to further provide an additional 3×-5× index compression ratio.

Through the Tensor-wise Bi-Random Sampling and compression, the final uploaded data of **each layer** are (*i*) the signs of the sampled tensors, each with multiple integer binary elements $\in \{-1, +1\}$; (*ii*) The compressed integer indices of corresponding tensors; (*iii*) A floating-point scaler $\mathcal{S}$.

F. Complexity Discussion

Given a DNN with N parameters, the time complexities of BIRD+'s three key steps, Layer Splitting, Bi-Random Sampling, and Tensors Compression are analyzed below:

1) *Layer Splitting*: Assuming a layer is split into k tensors, the L_1-Norm and L_∞-norm of each tensor will be calculated. The total time complexity is $O(k+N+N)$.
2) *Bi-Random Sampling*: L_1-Norm sampling and L_∞-norm sampling are two sequential random sampling processes. L_1-Norm sampling selects tensors from the total k tensors with a time complexity of $O(k)$. Assuming s tensors are randomly sampled, L_∞-norm-based uniform sampling further samples from the remaining s tensors, resulting in time complexity of $O(s)$.
3) *Tensors Compression*: since the scaler $\mathcal{S}$ has been calculated in the previous L_1-norm-based sampling, we only need to calculate signs for the remaining m tensors with time complexity of $O(d*m)$, where d is the number of elements in the sampled tensors.

Considering that $s < k < N$ and $d*m < N$, the overall time complexity of the above three steps can be simplified to $O(N)$. Furthermore, compared with the conventional element-wise sparsification-based method, the lightweight advantage of BIRD+ can be analyzed from the following two aspects.

1) The advantage of tensor-wise operation: Assuming that the operation objects in BIRD+ are k tensors with total e elements in a split layer. Obviously, the number of tensors is much smaller than the number of elements in a layer. Therefore, operating on tensors instead of elements not only reduces the time complexity from $O(e)$ to $O(k)$ but also lowers the I/O overheads of selected parameters.

To validate the above analysis, we conducted the following experiment: (1) we construct two vectors v_a and v_b with the same size of ten million floating-point numbers. (2) We split v_b into multiple chunks. (3) Element-wise random sampling is performed on v_a and tensor-wise random sampling is performed on v_b, respectively. (4) By sampling the same number of elements, we evaluate the time costs of the sampling process and I/O for the two sampling strategies. Figure 8 suggests that both the time costs of (a) the sampling process and (b) I/O in the tensor-wise random sampling are significantly lower than in element-wise random sampling. Additionally, the larger the block split, the lower the time costs.

2) The advantage of random sampling: the sorting-based Top-K algorithm is employed by many sparsification-based compressors. The time complexity of the popular sorting algorithm, which selects k maximum elements, is $O(Nlogk)$ [22], where N is the total number of parameters in the DNNs. In

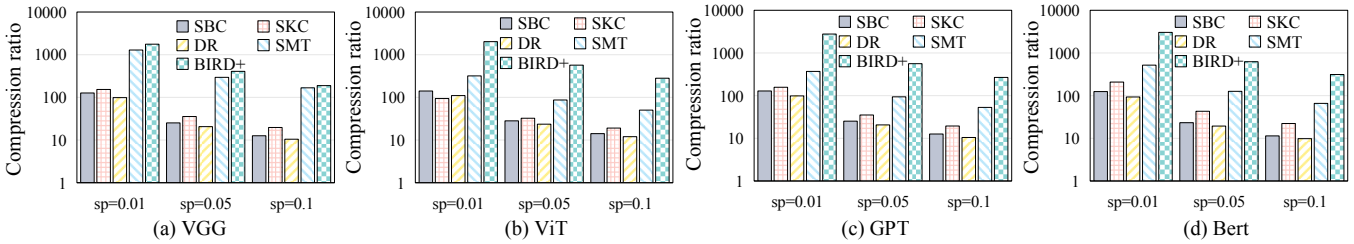


Fig. 9. Compression performance comparisons among different compression methods under the multiple sparsity levels.

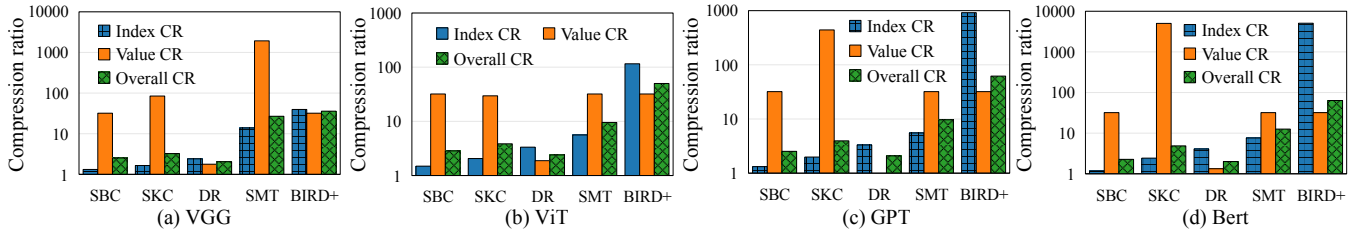


Fig. 10. The average index compression ratio (Index CR), value compression ratio (Value CR), and overall compression ratio (Overall CR) per selected parameter among different compression methods.

contrast, BIRD+ employs a random sampling mechanism by using the Monte Carlo simulation, resulting in a lower time complexity of $O(N)$.

V. Experiments

A. Experimental Setup

Tasks, Models, and Datasets: Our experimental workloads cover four mainstream distributed learning tasks of varying scales: ① Training VGG16 [34] model with 15.3M parameters on CIFAR10 [35]. The optimizer is Adam [37] with a learning rate of 1e-3. ② Training ViT-based [38] model with 0.91M parameters on Tiny ImageNet [39]. The optimizer is Adam [37] with a learning rate of 1e-3. ③ Training GPT-Mini [1] model with 13.9M parameters on WikiText2 [40]. The optimizer is Adam with a learning rate of 1e-2. ④ Training Bert-based [2] model with 67.1M parameters on Glue/MPRC [41]. The optimizer is Adam with a learning rate of 1e-2.

Baseline and Compared methods: In our study, we compared the performance of BIRD+ on the aforementioned tasks against five other state-of-the-art methods, which are sparsification-based distributed learning communication compression approaches: SBC [10], SKC [18], DR [33], SMT [15], and BIRD [42].

Experimental environment: In subsequent subsections, the experimental evaluations are conducted in a distributed learning environment comprising 10 clients and 1 server. During each communication round, clients initially train their local model for one epoch. Then, clients compress the updates of their local model using various compressors. Finally, the compressed model updates are uploaded to the server. The distributed training targets of VGG, ViT, GPT, and Bert are 85% validation accuracy,51% validation accuracy, 6.0 validation loss, and 70% validation accuracy, respectively (Note that these training targets are not aimed at achieving state-of-the-art model accuracy but are sufficient for evaluating the effectiveness and superiority among different approaches). The compression operations of each approach are implemented using the `Numpy v.1.19.1` [43] package in Python and executed on the xeon gold 6132 CPU. The communication module is built on the collective communication of `torch.distributed v.1.10.2` [44] package, utilizing the `Gloo` backend with 1 Gbps bandwidth.

B. Compression Ratio Under Different Sparsity

The compression ratio is one of the most crucial metrics for evaluating compression techniques. In this subsection, we compare the compression ratio of different compression approaches across four distributed learning tasks. For the sparsification-based communication compression technique, **sparsity (sp)** is a key factor (as defined in IV-D2). Sparsity reflects the number of elements *selected, compressed, and uploaded* by a compression method. The value of sparsity will influence the final communication compression ratio. Thus, this subsection evaluates and compares the overall communication compression ratio under the different sparsity levels for each compression approach.

Specifically, we set sparsity levels of 0.01, 0.05, and 0.1 for different compression methods by adjusting their hyper-parameters (e.g. k for Top-K sparsification-based compressors). Figure 9 suggests that *(i)* the overall compression ratios improve with the increase of sparsity. *(ii)* BIRD+ exhibits significant advantages over other sparsification-based methods under the different sparsity. Specifically, BIRD+ achieves a higher overall communication compression ratio of up to 32.6× compared to other compressors. Additionally, the sparsity of the adaptive compressor BIRD cannot be adjusted manually, thus its compression ratio cannot be evaluated under the different sparsity. In comparison, our proposed BIRD+ can achieve a higher compression ratio than BIRD by setting a lower sparsity (detailed evaluations are provided in subsection V-G). This controllable property of BIRD+ makes

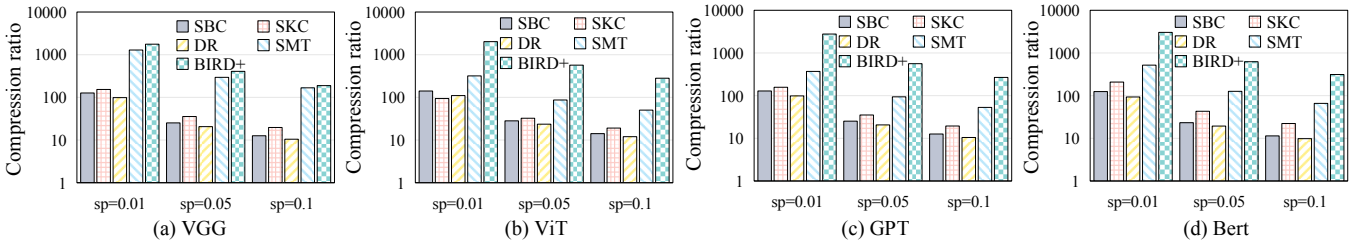


Fig. 11. Compression throughput (MB/s) comparisons among different compression methods under the multiple sparsity levels.

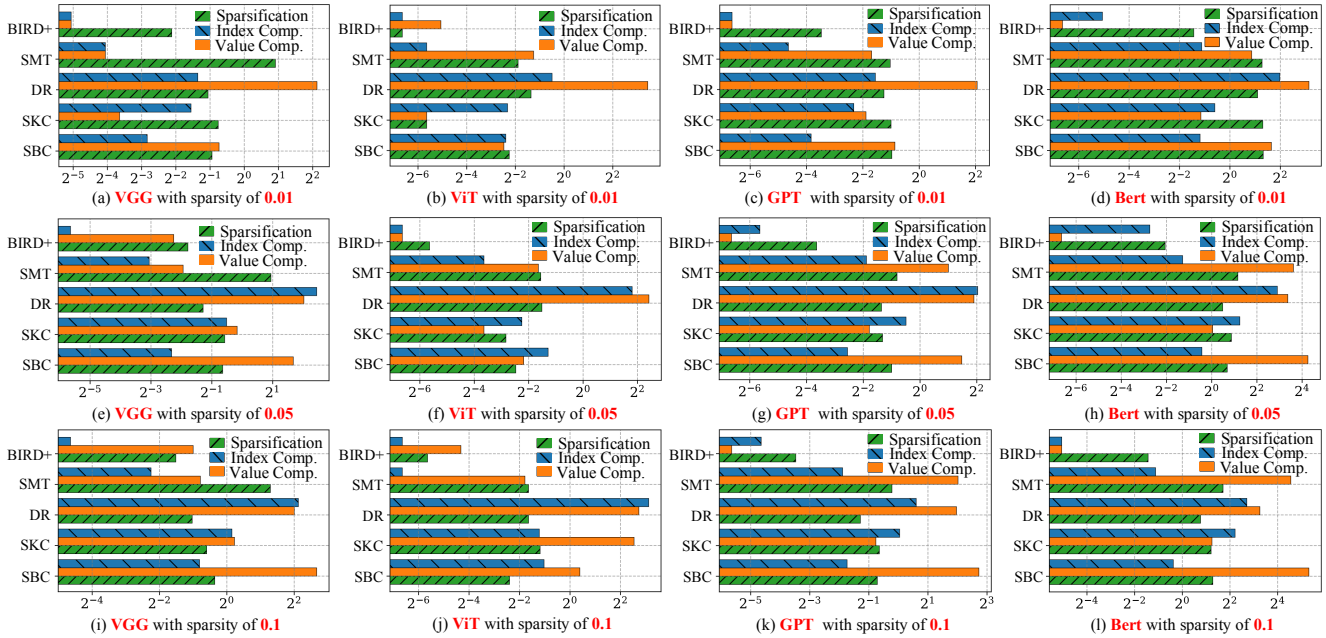


Fig. 12. The average compression times breakdown per distributed training round under the different sparsity levels.

it particularly advantageous for edge devices with limited network bandwidth and various user preferences.

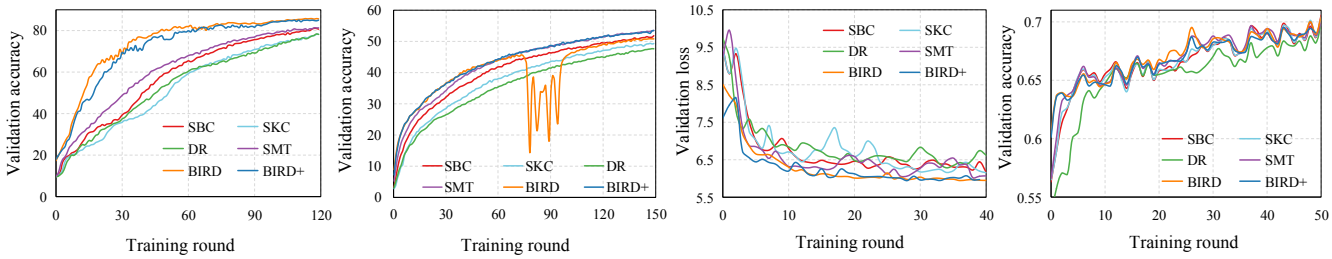
C. Compression Ratio Breakdown

To further investigate the advantages of BIRD+ in terms of compression ratio, we evaluate the average compression ratio breakdown per selected parameters, eliminating the influence of sparsity. Note that BIRD and BIRD+ have the same performance in this case. Figure 10 presents the average index compression ratio, value compression ratio, and the corresponding overall compression ratio **per selected parameter**. We observe that (*i*) SBC, SKC, and DR achieve a limited compression ratio ranging from 1.2×-4.1× on the index part by employing a lossless coding scheme for the massive indexes of all selected elements. Although SMT improves the index compression ratio in VGG16 by applying the index sharing technique to the 2-D convolution kernel of the convolutional neural network, it fails to address models without a convolutional structure (e.g., GPT and Bert). In contrast, BIRD+ introduces a tensor-wise index sharing mechanism, significantly reducing the proportion of indexes in different models, and then applies LZMA to a small number of shared indexes of these selected tensors, resulting in a substantially higher compression ratio ranging from 39.6×-5078.7× on the index part. (*iii*) Eq. 1 suggests that achieving high performance in either the index or value compression alone is insufficient. For example, although SKC achieves a considerable compression ratio from 29.6×-5024.5× on the value part, its overall communication compression ratio is still limited by its poor index compression ratio. In contrast, BIRD+ attains the highest overall compression ratio across all tasks with various DNN structures due to the considerable compression ratio achieved on both the index and value parts.

D. Throughput Under the Different Sparsity

As introduced in the subsection III, the primary purpose of the communication compression technique is to enhance the distributed training efficiency, making computational cost a crucial aspect of the compression method. This subsection evaluates and compares the computational costs of different compression approaches across four distributed learning tasks.

Considering that the numbers of parameters to be compressed vary under different sparsity, the total compression time will be influenced by the sparsity. Therefore, this subsection does not directly compare the overall compression times for each approach but defines a **Compression Throughput**



(a) Training VGG with the same cr=1020 (c) Training ViT with the same cr=120 (c) Training GPT with the same cr=330 (d) Training Bert with the same cr=140

Fig. 13. Model training quality comparisons among different methods under the same communication compression ratio.

(MB/s) to measure the computational costs of each method. Specifically, the compression throughput is calculated by $\frac{\alpha-\beta}{T}$), where α is the original model size, β is the compressed model size (i.e., the final communication costs), and T is the total time costs of the communication compression process. Compression throughput can effectively connect the compression ability and computational overheads, thereby reasonably reflecting the compression efficiency.

The experimental results in Figure 11 suggest that *(i)* the compression throughput of each method improves with the decrease of sparsity. This is because the compression times increase with the increase in data volume. *(ii)* BIRD+ achieves a higher compression throughput ranging from 2.98×-149.62× compared to other compressors. This is because BIRD+ has a lower compression computational overhead and a higher compression ratio than other methods under the same sparsity. *(iii)* Compared with BIRD+ which can achieve a higher sparsity by adjusting the hyper-parameter Gamma, BIRD adaptively achieves a relatively high sparsity in each task (0.017, 0.153, 0.097, 0.231 for VGG, ViT, GPT, and Bert, respectively). Thus BIRD+ has less data that needs to be compressed and the corresponding compression time.

E. Compression Times Breakdown

To study the advantages of BIRD+ in terms of computational efficiency, we divide the compression process of each method into three stages: sparsification, index compression, and value compression. Then we evaluate the time costs of each stage for different methods under various sparsity levels. Note that although BIRD cannot achieve different sparsity, BIRD and BIRD+ have the same compression times breakdown under the same sparsity theoretically.

Figure 12 illustrates that the time costs of each compression stage increase with the sparsity, while BIRD+ achieves the lowest time costs under the different sparsity levels. Specifically, sparsification times of BIRD+ only require about 0.05×-0.71× of compared methods because (*i*) the *Bi-Random sampling strategy* used in BIRD+ significantly reduces the time costs compared to the *element-wise sorting* used in other Top-K based compressors. (*ii*) The Tensor-based operation in BIRD+ incurs a lower I/O cost compared to operating on a large number of individual elements. Moreover, the index and value compression time of BIRD+ require 0.001-0.822× of compared methods because (*i*) the time costs of sign-based value quantization in BIRD+ are lower than the time-consuming sketching technique in SKC and DR; (*ii*) Unlike SBC, DR, and SMT, which spend a lot of time (about 60%-80%) on index lossless encoding, BIRD+ incurs almost negligible overheads in this aspect due to its index sharing mechanism, which generates only a small number of index data. Furthermore, the sparsification time of BIRD+ is slightly higher than BIRD, but the index and value compression time of BIRD+ is significantly lower than BIRD. This is because (*i*) BIRD+ has one more uniform sampling process in the sparsification process than BIRD; (*ii*) The index and value of sampled tensors that need to be compressed in BIRD+ are less than in BIRD under the lower sparsity. Therefore, BIRD+'s lightweight nature makes it particularly advantageous for edge devices with limited computational power, offering a higher compression performance at a lower computational cost.

F. Evaluation of Model quality

To demonstrate the convergence effectiveness of BIRD+'s sparsity-controllable tensor-wise random sampling, we compare the model quality of each method under the same communication compression ratio. Considering that the compression ratio of the preliminary work BIRD cannot be controlled by setting sparsity manually, we first run BIRD to obtain its adaptive compression ratio, and then assign the same compression ratio to other methods (including BIRD+) to match that of BIRD by adjusting their hyper-parameters.

Figure 13 illustrates that BIRD+ can achieve a similar or better convergence rate and accuracy compared to BIRD, and significantly better than other Top-K sparsification-based methods. This can be attributed to the fact that *(i)* under the limited communication traffic, BIRD and BIRD+ upload more values to the server, indicating that more 'important knowledge' learned from local data is transmitted to the server and improves the global model quality. *(ii)* Similar to the Top-K sparsification strategy, the Bi-Random sampling strategies in BIRD and BIRD+ also tend to select the elements with larger magnitudes in a more coarse-grained. *(iii)* The expectation invariance property of the random sampling in BIRD+ can effectively maintain the model training quality. It is worth noting that compared with the L_∞-norm-based second sampling of BIRD, the uniform sampling of BIRD+ can effectively avoid excessive magnitude of sampled elements, thereby achieving a more stable convergence rate. **More importantly, BIRD and**

TABLE I
PERFORMANCE COMPARISONS BETWEEN BIRD AND BIRD+.

Methods	Model Quality		Compression Ratio		Throughput(MB/s)	
	BIRD	BIRD+	BIRD	BIRD+	BIRD	BIRD+
VGG (Acc↑)	**85.75**%	85.37%	992.14	**1184.65**	313.90	**315.39**
ViT (Acc↑)	50.83%	**51.71**%	117.31	**138.66**	125.21	**152.61**
GPT (Loss↓)	**5.92**	5.94	261.56	**307.74**	349.52	**375.22**
Bert (Acc↑)	70.02%	**70.08**%	140.53	**1510.32**	262.87	**881.74**

BIRD+ achieve the above model training qualities without requiring extra memory costs to keep the unselected elements (i.e., the residual model). Therefore, BIRD+ can effectively utilize the limited bandwidth and memory resources to obtain a higher model training quality, which is crucial for bandwidth-constrained edge devices.

G. Comparisons between BIRD and BIRD+

The main differences between the preliminary compressor BIRD and our proposed BIRD+ are (1) BIRD+ introduces a controllable γ correction-based mechanism to flexibly balance the compression ratio and model training quality; (2) BIRD+ improves the convergence stability by developing a more effective uniform sampling strategy in the second sampling stage. To demonstrate the advantage of BIRD+, we first assign the appropriate hyper-parameter γ in BIRD+ to obtain the maximum communication compression ratio while ensuring that the model training quality is not significantly lower than the no compression baseline. Then we compare the model accuracy, communication compression ratio, and compression throughput of BIRD and BIRD+ (γ is set to 1.4, 1.75, 1.3, and 8.0 for the tasks of VGG, ViT, GPT, and Bert, respectively).

The experimental results in Table I suggest that *(i)* BIRD+ can obtain a similar model training quality as BIRD (↑ and ↓ are used to denote the higher is better in accuracy and the lower is better in Loss). This is because these two methods have a similar tensor sampling strategy. However, Figure 13 shows that BIRD+ can achieve a stabler convergence performance than BIRD. This is because uniform sampling of BIRD+ can effectively avoid excessive magnitude of the sampled elements, which helps maintain convergence efficiency. In contrast, the randomness of the training process causes BIRD to upload several elements with significant magnitude in the unpredictable training stage, potentially degrading the convergence stability during the training process. *(ii)* BIRD+ can improve the communication compression ratio of BIRD by up to 10.75× by setting a smaller sparsity using the controllable Gamma Correction mechanism. This advantage allows users to take full advantage of the limited bandwidth in resource-constrained devices. *(iii)* BIRD+ can improve the overall compression throughput of BIRD by up to 3.35× under the appropriate hyper-parameter γ. This is because although BIRD+ involves one more uniform sampling process than BIRD, the computational overhead of uniform sampling is not significant when the sampling object is tensor. Therefore, when the superiority in compression ratio is huge, such as the results in Bert, BIRD+ can obtain a much higher compression throughput than BIRD.

VI. CONCLUSION

In this paper, we propose BIRD+, a lightweight distributed learning communication compression algorithm, which applies a tenor-wise index sharing mechanism, a Bi-Random sampling strategy, and an expectation invariance property to solve the Low Index Compression Ratio issue, High Compression Computing Cost issue, and High Extra Memory Cost issue, respectively. Experimental results across multiple mainstream distributed learning tasks demonstrate that compared to the Top-K sparsification-based state-of-the-art distributed learning compression methods, BIRD+ achieves higher compression ratios up to 36.3× at lower time costs while maintaining the model quality without introducing the extra memory cost.

ACKNOWLEDGMENTS

This work was supported in part by the Major Key Project of PCL under Grant PCL2022A03, in part by Shenzhen Science and Technology Program under Grant RCYX20210609104510007 and Grant KJZD20230923114610021, in part by Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B1212010005, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2023A1515110072, in part by the National Natural Science Foundation of China under Grant Nos. 62032023 and T2125013, in part by the Innovation Funding of ICT, CAS under Grant No. E461050. (Corresponding author: Wen Xia).

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, and et al., "Language models are few-shot learners," in *Proc. NIPS'20*, 2020.
[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT'19*, 2019, pp. 4171–4186.
[3] H. Jin, D. Wu, S. Zhang, X. Zou, S. Jin, D. Tao, Q. Liao, and W. Xia, "Design of a quantization-based DNN delta compression framework for model snapshots and federated learning," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 3, pp. 923–937, 2023.
[4] M. Duan, D. Liu, X. Chen, and et al., "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," in *in Proc. ICCD'19*. IEEE, 2019, pp. 246–254.
[5] Z. Lian, J. Cao, Y. Zuo, and et al., "AGQFL: communication-efficient federated learning via automatic gradient quantization in edge heterogeneous systems," in *in proc. ICCD'21*. IEEE, 2021, pp. 551–558.
[6] X. Su, Y. Zhou, L. Cui, and J. Liu, "On model transmission strategies in federated learning with lossy communications," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 4, pp. 1173–1185, 2023.
[7] L. Cui, X. Su, Y. Zhou, and Y. Pan, "Slashing communication traffic in federated learning by transmitting clustered model updates," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2572–2589, 2021.
[8] L. Cui, X. Su, Y. Zhou, and J. Liu, "Optimal rate adaption in federated learning with compressed communications," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, May 2-5, 2022*. IEEE, 2022, pp. 1459–1468.
[9] D. Wu, W. Yang, H. Jin, X. Zou, W. Xia, and B. Fang, "Fedcomp: A federated learning compression framework for resource-constrained edge computing devices," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 43, no. 1, pp. 230–243, 2024.
[10] F. Sattler, S. Wiedemann, K. Müller, and et al., "Sparse binary compression: Towards distributed deep learning with minimal communication," in *Proc. IJCNN'19*, 2019, pp. 1–8.

[11] O. Goga and R. Teixeira, "Speed measurements of residential internet access," in *Proc. PAM'12*, ser. Lecture Notes in Computer Science, vol. 7192, 2012, pp. 168–178.
[12] K. Lee, J. Lee, Y. Yi, and et al., "Mobile data offloading: How much can wifi deliver?" *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 536–550, 2013.
[13] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
[14] S. U. Stich, J. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proc. NIPS'18*, 2018, pp. 4452–4463.
[15] D. Wu, X. Zou, S. Zhang, and et al, "Smartidx: Reducing communication cost in federated learning by exploiting the cnns structures," in *Proc. AAAI'22*, 2022, pp. 4254–4262.
[16] Y. Lin, S. Han, H. Mao, and et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. ICLR'18*, 2018.
[17] Aji and Heafield, "Sparse communication for distributed gradient descent," in *Proc. EMNLP'17*, 2017, pp. 440–445.
[18] J. Jiang, F. Fu, T. Yang, and et al., "Skcompress: compressing sparse and nonuniform gradient in distributed machine learning," *VLDB J.*, vol. 29, no. 5, pp. 945–972, 2020.
[19] S. Shi, X. Zhou, S. Song, and et al., "Towards scalable distributed training of deep learning on public cloud clusters," in *Proc. MLSys'21*, 2021.
[20] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Proc. NIPS'18*, 2018, pp. 1306–1316.
[21] D. Alistarh, D. Grubic, J. Li, and et al., "QSGD: communication-efficient SGD via gradient quantization and encoding," in *Proc. NIPS'17*, 2017, pp. 1709–1720.
[22] A. Mandal, H. Jiang, A. Shrivastava, and V. Sarkar, "Topkapi: Parallel and fast sketches for finding top-k frequent elements," in *Proc. NIPS'18*, 2018, pp. 10 921–10 931.
[23] W. Wen, C. Xu, F. Yan, and et al., "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. NIPS'17*, 2017, pp. 1509–1519.
[24] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS'17, 20-22 April 2017, Fort Lauderdale, FL, USA*, ser. Proceedings of Machine Learning Research, A. Singh and X. J. Zhu, Eds., vol. 54, 2017, pp. 1273–1282.
[25] F. Seide, H. Fu, J. Droppo, and et al., "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Proc. INTERSPEECH'15, Singapore, September 2014*, 2014.
[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *in Proc. CVPR'16, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.
[27] C. Chen, J. Choi, D. Brand, and et al., "Adacomp : Adaptive residual gradient compression for data-parallel distributed training," in *Proc. AAAI'18, New Orleans, LA, February 2018*.
[28] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and et al., "Error feedback fixes signsgd and other gradient compression schemes," in *in Proc. ICML'19, 9-15 June 2019, Long Beach, California, USA*, vol. 97, 2019, pp. 3252–3261.
[29] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.
[30] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *in Proc. INTERSPEECH'15, Dresden, Germany, September, 2015*.
[31] N. Dryden, T. Moon, S. Jacobs, and et al., "Communication quantization for data-parallel training of deep neural networks," in *Proceedings of 2nd Workshop on MLHPC, Salt Lake City, UT, November 2016*.
[32] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," in *Proc. ICLR'18, Vancouver, Canada, April 2018*.
[33] H. Xu, K. Kostopoulou, A. Dutta, and et al., "Deepreduce: A sparse-tensor communication framework for federated deep learning," *in Proc. NIPS'21*, vol. 34, pp. 21 150–21 163, 2021.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR'15, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
[35] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.
[36] E. Reinhard, G. Ward, and et al., *High Dynamic Range Imaging - Acquisition, Display, and Image-Based Lighting (2. ed.)*, 2010.
[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR'15*, 2015.
[38] A. Dosovitskiy, L. Beyer, A. Kolesnikov, and et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *in Proc. ICLR'21, Virtual Event, Austria, May 3-7, 2021*, 2021.
[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *in Proc. NIPS'12, December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012, pp. 1106–1114.
[40] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *Proc. ICLR'17*, 2017.
[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. ICLR'19*, 2019.
[42] D. Wu, W. Yang, X. Zou, and et al., "Bird: A lightweight and adaptive compressor for communication-efficient distributed learning using tensor-wise bi-random sampling," in *Proc. ICCD'23*, 2023, pp. 605–613.
[43] C. R. Harris, K. J. Millman, S. van der Walt, and et al., "Array programming with numpy," *Nat.*, vol. 585, pp. 357–362, 2020.
[44] A. Paszke, S. Gross, F. Massa, and et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. NIPS'19*, 2019, pp. 8024–8035.

VII. Biography Section

Donglei Wu is currently working toward the Ph.D. degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include Distributed Learning, Neural Network Compression. He has published several papers in major journals and international conferences including the TCAD, TPDS, TACO, AAAI and ICCD.

Weihao Yang is currently working toward the MS degree majoring in computer science at the Harbin Institute of Technology, Shenzhen, China. His research interests include Distributed Machine Learning, Neural Network Compression. He has published several papers in major journals and international conferences including the TCAD, TACO, and ICCD.

Xiangyu Zou (Student Member, IEEE) is an associate professor at the Harbin Institute of Technology, Shenzhen, China. His research interests include data deduplication, storage systems, etc. He has published several papers in major journals and international conferences including the TPDS, TOS, Future Generation Computing Systems, FAST, USENIX ATC, ICDE, MSST, ICPP, ICCD, AAAI, DAC, cluster, and HPCC. etc.

Hao Feng is a second-year PhD student in Intelligent Systems Engineering at Indiana University. He received his bachelor's degree in Computer Science and Technology from Shandong University in 2022. His research interests include High-Performance Computing, parallel computing, and recommender systems.

Dingwen Tao is a professor at the Institute of Computing Technology, Chinese Academy of Sciences, where he leads the High-Performance Data Processing Systems Group at the HPC Research Center. Previously, he served as a tenured associate professor at Indiana University. He obtained his Ph.D. in Computer Science from the University of California, Riverside in 2018, following his B.S. in Mathematics from the University of Science and Technology of China in 2013. Dr. Tao has been recognized with several prestigious awards, including the 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 High Performance Computing (2020), NSF CRII Award (2020), and IEEE CLUSTER Best Paper Award (2018). Currently, he serves as an Associate Editor for the IEEE Transactions on Parallel and Distributed Systems and Parallel Computing Journal. He has also chaired the IEEE ScalCom-2021, DRBSD workshops, and IWBDR workshops. Dr. Tao is a Senior Member of both ACM and IEEE.

Shiyi Li received the BS degree in mathematics from Northwest University, Xi'an, China, in 2009, and the PhD degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently a research associate with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen. His research interests include dependable storage systems and coding theory, cloud storage, etc. He has published more than 20 papers in prestigious international conferences and journals, such as the TPDS, TACO, USENIX ATC, ICDE, SRDS, ICPP, ICCD, etc.

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 a professor at 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 50 papers in major journals and conferences including the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, Proceedings of the IEEE, USENIX ATC, FAST, DAC, ICDE, AAAI, HotStorage, MSST, DCC, IPDPS, ICPP, ICCD, etc.

Binxing Fang Binxing Fang received his Ph.D. from Harbin Institute of Technology, China in 1989. He is a member of the Chinese Academy of Engineering and a professor in the School of Harbin Institute of Technology, Shenzhen, China. He is currently the chief scientist of the State Key Development Program of Basic Research of China. His current interests include big data and information security.