DRAGONN: Distributed Randomized Approximate Gradients of Neural Networks

Zhuang Wang [*] [1] **Zhaozhuo Xu** [*] [1] **Xinyu Crystal Wu** [1] **Anshumali Shrivastava** [1] [2] **T. S. Eugene Ng** [1]

Abstract

Data-parallel distributed training (DDT) has become the de-facto standard for accelerating the training of most deep learning tasks on massively parallel hardware. In the DDT paradigm, the communication overhead of gradient synchronization is the major efficiency bottleneck. A widely adopted approach to tackle this issue is gradient sparsification (GS). However, the current GS methods introduce significant new overhead in compressing the gradients, outweighing the communication overhead and becoming the new efficiency bottleneck. In this paper, we propose DRAGONN, a randomized hashing algorithm for GS in DDT. DRAGONN can significantly reduce the compression time by up to 70% compared to state-of-the-art GS approaches, and achieve up to $3.52\times$ speedup in total training throughput.

1. Introduction

Recent years have witnessed the popularity of Deep Learning (DL) in both research and industrial communities. Deep Neural Networks (DNN) trained on massive amount of data have experienced significant success in computer vision (Dosovitskiy et al., 2020; Tolstikhin et al., 2021), language processing (Vaswani et al., 2017; Radford et al., 2019), recommendation systems (Naumov et al., 2019) and geophysics (Desai et al., 2021).

The Need for Data-parallel Distributed Training: Recently, multilayer perceptron (MLP) models have demonstrated improvement over state-of-the-art (SOTA) DL benchmarks (Tolstikhin et al., 2021; Liu et al., 2021). However, these MLP models introduce million-scale weight tensors. As a result, large batch training over these giant weight tensors is infeasible due to the limited memory resources of Graphic Processing Unit (GPU) (Owens et al., 2008). To overcome this obstacle, data-parallel distributed training (DDT) becomes a widely adopted paradigm. In this paradigm, the data loader partitions the dataset according to the number of workers. Next, each worker only trains on its own partition. Then, the model gradients generated from each worker are synchronized. Finally, the weights at each worker is updated by the synchronized gradients. This data parallelism can scale the training of neural network over large numbers of GPUs with reduced total training time.

Communication Bottleneck in Gradient Synchronization: The innovations of hardware accelerators (Luo et al., 2018; NVIDIA, 2021) and domain-specific software optimization (Chen et al., 2018; Zheng et al., 2020; Chetlur et al., 2014) have dramatically reduced the iteration time of DNN training jobs. This trend results in more frequent gradient synchronization in DDT. However, it is difficult for the cloud network bandwidth to keep up with the pace of the computation-related improvement (Sapio et al., 2019; Bai et al., 2021). Moreover, the number of GPUs for DNN training keeps increasing due to the ever-growing training dataset, which further worsens the communication time (Jiang et al., 2020; Thakur et al., 2005). The communication overhead in DDT has become a well-known performance bottleneck as each GPU needs to transmit the full gradients for synchronization (Fei et al., 2021; Huang et al., 2019; Aji & Heafield, 2017; Lin et al., 2017).

Gradient Sparsification for Efficient Communication: Gradient sparsification (GS) (Strom, 2015; Aji & Heafield, 2017; Shi et al., 2021; Stich et al., 2018; Barnes et al., 2020) has great potential to alleviate the communication bottleneck. GS approaches only select a subset of the original gradients for synchronization. There exists two well-known GS paradigms, namely exact TopK (Aji & Heafield, 2017) and approximate TopK, such as deep gradient compression (DGC) (Lin et al., 2017) and MSTopk (Shi et al., 2021). The exact TopK method selects the exact top-k gradients while approximate TopK selects the gradients with values greater than an estimated threshold. Both approaches can save up to 99.9% of the gradient exchange in DDT while preserving the iteration-wise performance towards convergence. Approximate TopK can achieve higher practical training speed

[*]Equal contribution [1]Computer Science Department, Rice University, Houston, TX, USA [2]ThirdAI Corp, Houston, TX, USA. Correspondence to: Anshumali Shrivastava <anshumali@rice.edu>, T. S. Eugene Ng <eugeneng@cs.rice.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

than exact TopK because of the reduced overhead of sorting operations.

Overhead in Gradient Sparsification: Unfortunately, we have seen minimal practical benefits of the GS algorithms (Agarwal et al., 2021; Xu et al., 2021). The main reason comes from the overhead in compressing the gradients. The SOTA GS system (Lin et al., 2017) requires *exact* operations that extract *approximate* top gradients and write them into memory for communication. The algorithmic level design of such operations leads to significant time costs that limit the efficiency improvement of GS systems. So far, there have been few demonstrations that the current GS in any form can achieve the expected total acceleration over full gradient synchronization.

Given the efficiency bottleneck of GS algorithms in DDT, it is natural to ask the following question.

Is there any technique that can overcome the compression overhead of GS in DDT while maintaining the iteration wise convergence?

In this paper, we answer this question by proposing DRAGONN, a randomized hashing algorithm for GS. We argue that it is unnecessary to use exact operations on GPUs (i.e., parallel prefix sum (Blelloch, 1990; Harris et al., 2007)) because only approximate top-k gradients are required. To resolve this exact-approximate mismatch, DRAGONN provides a randomized hashing algorithm that directly assigns memory locations for each value in the approximate top-k gradients. In this way, DRAGONN naturally supports massively parallel gradient extraction as multiple threads can write the gradients simultaneously. Moreover, we perform a series of system-level optimizations to maximize the efficiency of DRAGONN.

Our Contributions: The main contributions of this paper are summarized as follows:

- We propose DRAGONN, a randomized hashing algorithm for GS in the DDT of neural networks. Through the hashing algorithm, we significantly reduce the compression overhead in GS while preserving the iteration wise accuracy. We also upper bound its compression error and provide a theoretical analysis on the generalization error of the models trained via DRAGONN.
- We perform system-level optimizations of DRAGONN. We first introduce efficiency-aware tensor selection over the neural network so that we only perform GS on tensors leading to overall efficiency improvement. We also introduce a sparse decoding method to ensure that the decoding overhead does not increase linearly with the number of GPUs.
- Our extensive evaluation in vision and recommendation shows that to reach the same level of convergence, DRAGONN achieves up to $3.52\times$ speedup in total training time over DGC. Moreover, we provide detailed micro-benchmarks that suggest DRAGONN saves up to 70% of the compression time of current GS methods while reducing the decoding overhead from linear with the number of GPUs to nearly constant.

2. Background and Motivation

2.1. Data-parallel Distributed Training

Data-intensive training of neural networks (NN) on powerful Graphic Processing Unit (GPU) (Owens et al., 2008) boosts the success of Deep Leaning (DL) (LeCun et al., 2015). Given massive amount of training examples, data-parallel distributed training (DDT) (Shallue et al., 2019; Ben-Nun & Hoefler, 2019; Li et al., 2020) has become one of the most popular paradigms to scale out DL with multiple GPUs. In this paradigm, the training set is partitioned into multiple subsets. Each GPU has a replica of the training model that trains on a specific subset. In each iteration, each GPU consumes a mini-batch from its allocated subset as the input of the training. Next, it propagates the mini-batch through the NN model and calculates the loss function via *forward propagation*. Then, it uses the loss value to compute the gradients of each parameter in *backward propagation*. Finally, it synchronizes the gradient updates from all GPUs to update the model parameters with a certain optimizer, such as SGD (Zinkevich et al., 2010) and Adam (Kingma & Ba, 2014). Training a DNN model is a process to refine the model parameters with the above steps iteratively until its convergence. In this paper, we focus on the synchronous DDT because of its wide adoption (Abadi et al., 2016; Jiang et al., 2020; Sergeev & Del Balso, 2018; Li et al., 2020).

2.2. Communication Bottleneck in DDT

The single-GPU iteration time of DNN training jobs has been significantly reduced thanks to the advancement of both DNN accelerators and domain-specific compiler techniques. For example, the iteration time of ResNet50 with one GPU has decreased by $22\times$ in the last six years (Sun et al., 2019). However, the cloud network bandwidth has only seen a roughly $10\times$ increase in the same period (Zhou et al., 2020). This imbalance between the fast-growing computing capability and the slower-growing communication bandwidth worsens the communication-computation tense in DDT. It has been reported that the communication time for gradient synchronization accounts for over 60% of the total time for the training of BERT (Devlin et al., 2018) or other Transformer models across 16 AWS EC2 instances, each with 8 NVIDIA V100 GPUs, in a 100Gbps network (Bai et al., 2021). We denote the *total overhead* as the cumulative time spent in procedures other than forward and backward propagation in the DDT of neural networks.

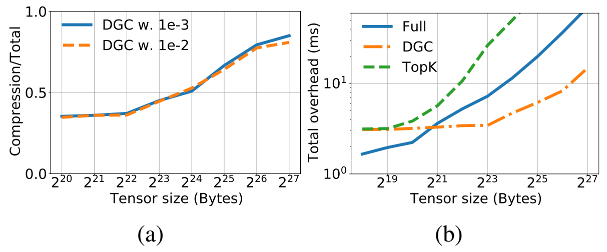


Figure 1. (a) the ratio between compression overhead and total overhead in different tensor size (bytes). (b) The total overhead in data-parallel distributed training versus tensor size (bytes).

Table 1. The number of operations in existing approximate TopK and hashing-based gradient compression algorithms to get the indices of the top gradients on CPUs and GPUs. d and l are the number of gradients and of gradients greater than the threshold. (**comp.** compare)

GS	CPU	GPU
Threshold	d **comp.**	$2d$ **comp.**,$2d-2$ **add.**, $d-1$ **swap.**
Hashing	d **comp.**, l **hash.**	d **comp.**, l **hash.**
Lower bound	d **comp.**	d **comp.**

2.3. The Need for Approximate TopK GS

In current GS approaches, exact TopK (Aji & Heafield, 2017) selects the exact top-k gradients at the cost of prohibitive compression overhead. Deep gradient compression (DGC) (Lin et al., 2017) achieves the SOTA performance in DDT. It first estimates the kth largest absolute value from a subset where each gradient in this set is uniformly sampled from the gradient tensor. Next, DGC uses this approximate value as a threshold. Only gradients with absolute values greater than this threshold is selected for synchronization, as shown in Algorithm 3. The main reason for DGC's outstanding performance is that the approximate TopK operation balances the trade-offs between accuracy and efficiency. It preserves the accuracy by synchronizing gradients with large absolute values while reduces the compression overhead in exact sorting. Therefore, it stands out as a major benchmark in GS.

2.4. The Limitations of Previous Approximate GS

Our experimental observation shows that DGC still introduces costly compression overhead, which degrades the training performance of DDT in practice. For instance, if we perform DGC for vision transformer (Dosovitskiy et al., 2020) on 16 Nvidia V100 GPUs with 0.001 compression ratio in a 25Gbps network, the expected speedup over full gradient synchronization is $2.81\times$. However, in practice, we could only observe $2.14\times$ speedup.

To further evaluate the impact of compression procedure in DGC, we study the ratio between the compression overhead and the total overhead in the DDT of a tensor over 16 Nvidia V100 GPUs. We present this ratio versus the tensor size in Figure 1(a) with compression ratio 0.001 and 0.01. From the figure, we observe that when the tensor size exceeds 2^{24} bytes, the compression overhead outweighs other overheads such as communication overhead and becomes the major efficiency bottleneck. The communication time has been significantly reduced and it can be close to the communication latency after compression. However, the compression overhead increases with the tensor size. So does the gap between the communication time and compression overhead. Because compression competes for GPU resources with backward propagation, it delays the training and thus degrades the performance.

Moreover, for smaller tensor that compression overhead does not exceed 50% of the total overhead, the existence of compression procedure still degrades the practical performance. To demonstrate this, we study the total overhead of full gradient synchronization and DGC as the tensor size increases when we communicate the tensor over 16 Nvidia V100 GPUs via compression ratio 0.001. Figure 1(b) shows that for tensor size smaller than 2^{21} bytes, the full gradient synchronization outperforms DGC in efficiency even if DGC can significantly save the network traffic volume. It is because there is a constant overhead to launch GPU kernels for compression (Sergeev & Del Balso, 2018; Wang et al., 2021) and it is even greater than the saved communication time for small tensors. The exact TopK always performs worse than the full gradient synchronization.

To fully unleash the benefits of GS for DDT, we must design efficient compression algorithms to minimize the compression overhead while preserving the performance in convergence. In addition, we must carefully enable compression for tensors to avoid the over-compression penalties.

3. DRAGONN: A Hashing-based Compressor

3.1. The Inefficiencies of Previous Approximate GS

Although DGC (Lin et al., 2017) can reduce the overhead of sorting operations, its compression overhead still greatly dilutes the benefits gained from communication time savings. The performance bottleneck of DGC lies in the process of extracting the indices of the top gradients (Lines 2-3 in Algorithm 3, Appendix B). When the compression operation is performed on CPUs, it needs d comparison operations to determine the gradients greater than the threshold, where d is the number of gradients in a tensor [1]. Note that this is the lower bound of the number of operations to extract the top gradients. It then writes the selected gradients along

[1]There is no need to have the mask with CPU compression.

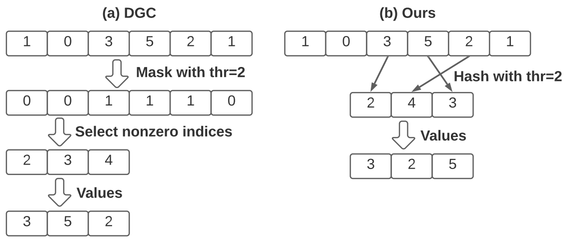


Figure 2. (a) The DGC algorithm for extracting top gradients and write it into the memory (b) DRAGONN. We overcome the overhead of nonzero indices selection by direct hashing.

Algorithm 1 DRAGONN

```
Input: gradient $\mathcal{G}$, threshold $t$, memory size $m$, hash function $h$
Output: $\mathcal{G}_c$, $I$
$d \leftarrow \mathsf{len}(\mathcal{G})$
$I \leftarrow \mathbf{-1}_m$
for $i = 0$ to $d - 1$ do
    if $|\mathcal{G}[i]| \geq t$ then
        $j \leftarrow h(i)$
        $I[j] \leftarrow i$
    end if
end for
$\mathcal{G}_c = \mathcal{G}[I]$
```

with their indices into the memory sequentially. GPUs are widely used to speed up compression operations (Xu et al., 2021). Parallel prefix sum (Harris et al., 2007) is the SOTA algorithm to select nonzero indices to accelerate this process (Meta, 2022; Google, 2022). In order to avoid the sequential memory writing issue, it performs $2d$ comparison, $2(d-1)$ add , and $d-1$ swap operations, resulting in scanning the tensor multiple times and incurring non-negligible overheads (refer to Appendix C). The numbers of operations to get the indices of the top gradients are in Table 1.

The number of memory access in DGC's compression operations with GPUs is around $7\times$ higher than the lower bound [2]. Moreover, parallel prefix sum builds a balanced binary tree for parallel computing, leading to $O(\log d)$ steps in a sequence. These two factors explain the high compression overheads of DGC (and other existing approximate GS algorithms) observed in Section 2.4.

Unnecessary Exact Algorithm for Approximate Gradient Compression: DGC is an approximate gradient compression algorithm, but its performance bottleneck lies in parallel prefix sum, which is an exact algorithm. We argue that replacing parallel prefix sum with an approximate algorithm does not affect the performance in the convergence of GS in DDT. Moreover, the approximate algorithm could significantly reduce the compression overhead.

3.2. DRAGONN

We design DRAGONN, a hashing-based compression algorithm to minimize the compression overhead, as shown in Algorithm 1. It allocates memory for the indices in advance according to the compression rate. It first compares the gradient values with the threshold. If a gradient (or its absolute value) is no less than the threshold, its index is mapped into an integer within a range, i.e., from 0 to $m-1$. It then writes the index into the memory with the offset of the mapped integer. If multiple indices are mapped into the same integer, they are written into the same position and the previous index is overwritten by the latter ones. After this step, the indices of the top gradients are extracted. It then obtains the gradients based on the indices. An index -1 indicates that there is no index written into this position. DRAGONN sets this index and the corresponding gradient to 0. Figure 2 compares DGC and DRAGONN. The difference lies in how they extract the indices of the top gradients.

Algorithm Complexity. DRAGONN has d comparison operations and l hashing operations, where l is the number of gradients greater than the threshold, to get the indices of the top gradients with both CPUs and GPUs for compression, as shown in Table 1. In addition, unlike parallel prefix sum, there is no dependency among the hashing operations. The lower bound of the overhead is d comparison operations as it has to at least scan and compare all the gradients once. Therefore, the hashing-based GS can achieve the near-optimal performance for approximate TopK gradient compression.

Strength in Parallel Computing. The critical path in approximate GS is the sequential memory writing for the indices of top gradients (Line 14 in Algorithm 3). In contrast, DRAGONN writes an index into the position based on the mapped integer and it allows for hash collisions (refer to Section A.2 for the analysis). Because memory writing is an atomic operation on GPUs (Owens et al., 2008), it guarantees that there is exact one valid index in the position when hash collisions occur. Multiple threads can perform the hash functions and memory writing simultaneously without extra operations. An interesting property of DRAGONN is that, unlikely DGC, the indices in the memory are in random orders. Because gradients are independent with each other in the memory, the order of them does not affect the results and there is no need to sort them.

Moreover, we provide a theoretical analysis on the convergence and generalization of DRAGONN in Appendix A.

4. Deploying DRAGONN in Practice

DRAGONN is a tensor wise approximate GS with near-optimal compression overhead. As a DNN model typi-

[2]There are two memory accesses in add and swap operations.

Algorithm 2 DDT via DRAGONN

```
Input: weights $\theta$, learning rate $\eta$, number of gradients $d$
Output: $\tilde{g}$
$g :=$ stochastic_gradient($\theta$)
if $T_{\mathsf{comp}}(d) < T_{\mathsf{full}}(d) - T_{\mathsf{spr}}(d)$ then // Eq.(1)
    $g_c :=$ DRAGONN$(g)$
    $g_m :=$ communicate$(g_c)$
    $\tilde{g} :=$ sparse_decode$(g_m)$
else
    $\tilde{g} =$ allreduce$(g)$
end if
```

cally consists of multiple tensors, a practical challenge we must address is how to deploy and optimize DRAGONN in DDT for better efficiency improvement. In this section, we present the system-level optimizations for the performance of DRAGONN in practice. We start with an overview of components in the GS based DDT system via DRAGONN. Then, we introduce the efficiency-aware tensor selection. Finally, we introduce the sparse decoding technique to improve the aggregation efficiency.

4.1. Overview

Algorithm 2 describes the DDT via DRAGONN. Given a tensor with d gradients, DRAGONN first determines whether GS has benefits based on a cost-benefit analysis considering both the system configuration and compression algorithm (Section 4.2). If there are no benefits, it directly synchronizes the tensor with Allreduce without compression. Otherwise, it compresses the tensor via DRAGONN and resorts to other collective communication operations (e.g., Allgather) for the communication. Each worker then receives the compressed tensors and decodes them with the proposed sparse decoding mechanism (Section 4.3).

4.2. Efficiency-aware Tensor Selection

A strawman to deploy DRAGONN in DDT is to apply it on all tensors in a DNN model (Xu et al., 2021; Shi et al., 2021). Unfortunately, it incurs over-compression penalties and harms overall performance (Bai et al., 2021). In the DDT of a DNN model with multiple tensors, the communication time is mainly contributed by large tensors and small tensors are not the culprit of the poor scalability. Because the communication latency can dominate the communication time for small messages (Nvidia, 2018; Sergeev & Del Balso, 2018; Li et al., 2020), there are no benefits to further reduce the size of small tensors due to the communication latency. Moreover, compression incurs computational overheads and competes for GPU resources with training. Hence, there is no guarantee that compressing tensors can always improve the training performance.

We develop a general cost-benefit analysis to determine the set of tensors that should be compressed to avoid over-compression penalties. We must ensure that the compression time is smaller than the communication time savings:

$$T_{\mathsf{comp}}(d) < T_{\mathsf{full}}(d) - T_{\mathsf{spr}}(d), \tag{1}$$

where d is the number of gradients in a tensor, $T_{\mathsf{comp}}(d)$ is the compression time, $T_{\mathsf{full}}(d)$ and $T_{\mathsf{spr}}(d)$ are the communication time without and with compression, respectively.

Suppose the number of workers involved in the training is K, the network bandwidth is B, and the size of each gradient is A. The communication cost models follow the analysis in the literature (Patarasuk & Yuan, 2009; Thakur et al., 2005; Fei et al., 2021). There are many available communication schemes for the gradient synchronization (Jiang et al., 2020; Sergeev & Del Balso, 2018; Xu et al., 2021; Bai et al., 2021), while we take Ring Allreduce and Allgather as the examples to quantify the cost-benefit analysis.

Ring Allreduce is a widely-adopted Allreduce algorithm for gradient synchronization in DDT without compression (Li et al., 2020; Sergeev & Del Balso, 2018). The communication time for a tensor with the size of dA is

$$T(d) = 2(K-1)(\alpha + \frac{dA}{KB}),$$

where α is the one-way network latency between workers.

However, the aggregation operations of compressed gradients are not associative, which makes compressed tensors not allreducible (Agarwal et al., 2021). Allgather is commonly used for the gradient synchronization with gradient compression (Xu et al., 2021; Wang et al., 2021). Hence, the communication time after compression becomes

$$T_{\mathsf{comp}}(d) = (K-1)(\alpha + \frac{\gamma dA}{B}),$$

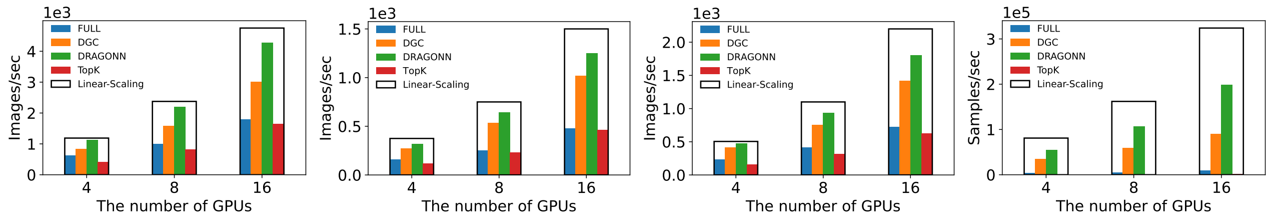
where γdA is the tensor size after compression. Therefore, we show that

$$T_{\mathsf{full}}(d) - T_{\mathsf{spr}}(d) = (K-1)(\alpha + (1/K - \gamma)dA/B)$$

Note that other collective communication operations can also be applied to this cost-model analysis.

The compression time consists of two parts: encoding time and decoding time. Given a compression algorithm and the compression ratio, we can easily profile both the encoding and decoding time offline as a function of the number of gradients in a tensor. We then have the cost model for the compression time $T_{\mathsf{comp}}(d)$.

We compare $T_{\mathsf{comp}}(d)$ and $T_{\mathsf{full}}(d) - T_{\mathsf{spr}}(d)$ to decide whether it has benefits to enable compression for a tensor. The decision depends on the tensor size, the compression algorithm, the compression ratio, the number of workers, the network bandwidth, and the computing capacity.



(a) ResNet50 (b) ViT (c) MLP-Mixer (d) XML

Figure 3. The DDT speed with different number of GPUs via different GS approaches for four DNN models.

Table 2. Overall Training speedups of DRAGONN over DGC and FULL when reaching 94% accuracy in Cifar10 and 80% accuracy in Wiki10-31K.

Speedups	ResNet50	ViT	MLP-Mixer	XML
DGC	1.42×	2.15×	1.72×	3.52×
FULL	2.40×	2.45×	6.04×	35.9×

4.3. Sparse Decoding

Because the compressed tensors are not allreducible, each worker receives K compressed tensors after communication and they must be decoded before the weight updates. To handle this, current GS approaches first decode each compressed tensor into a dense format and then perform aggregation operations (dense decoding) (Xu et al., 2021; Wang et al., 2021). This procedure generates a sparse-to-dense overhead that linearly increases with the number of GPUs. As the sparse-to-dense conversion and aggregation operations are non-negligible on GPUs, the decoding procedure could greatly degrade the final performance.

We design a sparse decoding mechanism to reduce the decoding time. Because the index-value pairs in compressed tensors are independent from each other, we batch these K compressed tensors, i.e., concatenate all of them, for one decoding operation. After decoding, there is only one dense tensor so that we need no aggregation operations. This batching mechanism is also applicable to other GS algorithms, such as DGC, TopK, and MSTopk (Shi et al., 2021). The sparse decoding is equivalent to dense decoding and it can support the indices in any order.

5. Experiment

In this section, we demonstrate the efficiency of our approach in DDT. We start with presenting the testbed. Next, we present an evaluation of our approach over end-to-end training. Finally, we showcase several micro-benchmarks as an ablation study.

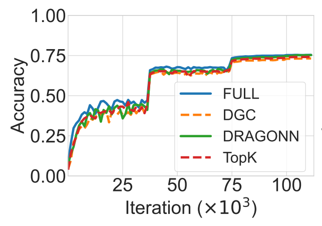
Testbed: We perform experiments on 16 Nvidia Tesla V100-32GB GPUs. Each machine has 8 GPUs, 96 CPU cores (Intel Xeon 8260 at 2.40GHz) and 256 GB of RAM. The GPU-to-GPU interconnection is supported by PCIE and the network bandwidth connecting machines is 25Gbps. The machines run Debian 10 operating system and the software environment includes CUDA 11.0, PyTorch-1.8.0, NCCL-2.7.8., and Hovorod-0.19.1. Memory momentum correction (Lin et al., 2017) is used as the error-feedback mechanism to preserve the training accuracy of all the evaluated scarification algorithms.

5.1. Main Results

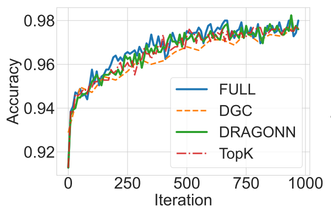
We present the results that demonstrate our method's advantage in neural network training over DGC (Lin et al., 2017) and TopK (Aji & Heafield, 2017). Specifically, we would like to answer the following questions: (1) What are the total speedups of DRAGONN over DGC and TopK? (2) Does DRAGONN preserve the iteration wise convergence in test accuracy as full synchronization (FULL)?

Settings: We compare the performance of DRAGONN against DGC, TopK and full synchronization in the data-parallel distributed training of four deep models. For visual recognition, we choose ResNet50 (He et al., 2016), Vision Transformer (ViT) (Dosovitskiy et al., 2020) and MLP-Mixer (Tolstikhin et al., 2021) for study. The input image size is set to $224 \times 224 \times 3$ for all these models. We set the evaluation metric to be the accuracy. We evaluate DRAGONN and baseline methods on the DDT of ResNet50 over ImageNet-1K (Deng et al., 2009) dataset. We use Adam (Kingma & Ba, 2014) as optimizer with batch size 64 and learning rate 0.1. After linearly warming up the learning rate, we reduce by 10 on the 30th, 60th and 80th epochs. We also evaluate DRAGONN and baseline methods over ViT and MLP-Mixer on fine-tuning tasks: given weights pretrained on ImageNet-21k (Ridnik et al., 2021), we perform DDT on Cifar10 (Krizhevsky et al., 2009). We set Adam (Kingma & Ba, 2014) as optimizer with learning rate 10^{-5} and batch size 32.

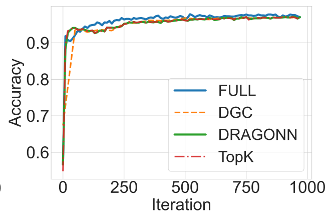
Moreover, we compare the performance of our method against DGC for DDT over giant weight tensors. We choose multi-label classification to showcase this comparison. We use the Wiki10-31K dataset in the extreme classification repository (Bhatia et al., 2016). There are 14146 training samples and 6616 test samples. We embed the input fea-



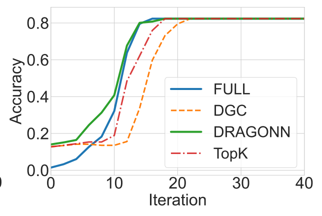
(a) ResNet50



(b) ViT

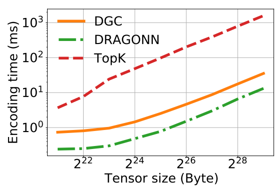


(c) MLP-Mixer

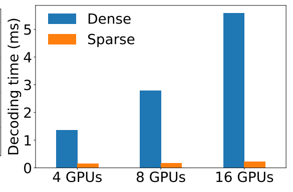


(d) XML

Figure 4. Convergence comparison of four DNN models with different GS.



(a)



(b)

Figure 5. (a) the encoding time of various compression algorithms in different tensor sizes (bytes) (b) the decoding time of dense decoding and sparse decoding.

tures following (Chen et al., 2020) and only train on the last linear layer. The last layer has input 4096 and outputs 30938. As we set the linear layer without bias, we perform distributed training over a single giant tensor with 126.72M parameters. The batch size is 256. We use learning rate 0.001 for Wiki10-31K. For evaluation, we denote accuracy as the $P@1$ (top-1 accuracy) as our metrics. We denote the model as XML.

Speedups: we present the DDT speed on 4, 8 and 16 GPUs of DRAGONN and baselines in Figure 3. As shown in the figure, when we perform DDT over 16 GPUs, DRAGONN performs $2.40\times$ speedup over full synchronization in ResNet50, while DGC performs $1.68\times$ speedup. DRAGONN performs $2.61\times$ speedup over full synchronization in ViT, while DGC performs $2.14\times$ speedup. DRAGONN performs $2.57\times$ speedup over full synchronization in MLP-Mixer, while DGC performs $2.06\times$ speedup. DRAGONN performs $22.4\times$ speedup over full synchronization in XML, while DGC performs $10.2\times$ speedup. Meanwhile, TopK (Aji & Heafield, 2017) introduces large overhead in compression and has the slowest speed. These results answer the first question: DRAGONN achieves significant speedups over baselines in various tasks.

Convergence: We present the iteration wise convergence of DDT with four models in Figure 4(b). We observe that both DRAGONN and TopK approximate the full synchronization better than DGC. For XML, it takes even $1.6\times$ more iterations for DGC than DRAGONN to converge. Combining the iteration wise performance and time wise speedup on 16 GPUs, we summarize the total time speedup of DRAGONN over baselines in Table 2. These results indicate that DRAGONN achieves at most $3.52\times$ speedup over DGC and $35.9\times$ speedup over full synchronization.

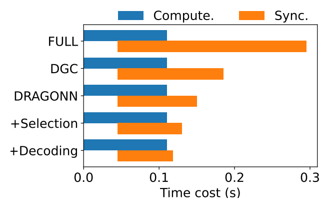
Discussion: The speedup of DRAGONN over DGC increases with the number of GPUs. This phenomenon demonstrates DRAGONN's strength in scalable DDT. Moreover, we expect that DRAGONN will become more beneficial in faster networks (e.g., NVLink for the GPU-to-GPU interconnection or 100Gbps network bandwidth) because the ratio of communication to compression overheads shifts to the latter. Furthermore, the performance improvements of MLP-Mixer and ViT are consistent as long as the input images have the same shape. Therefore, the advantages of DRAGONN are further validated.

5.2. Micro-benchmarks

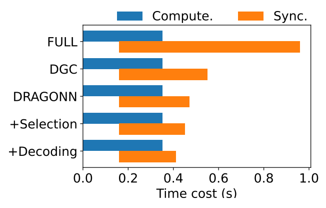
In this section, we would like to investigate the components in DRAGONN that help its speedup in DDT. We evaluate the effectiveness of DRAGONN's three components: 1) the hashing-based compressor, 2) efficiency-aware tensor selection, and 3) sparse decoding.

Figure 5(a) compares the encoding time of TopK (Aji & Heafield, 2017), DGC, and DRAGONN over various tensor sizes with the compression ratio 0.001. We can see that exact TopK algorithm performs much worse than approximate TopK algorithms. Therefore, we only provide the comparison of DRAGONN with DGC. DRAGONN has up to 70% lower encoding time than DGC thanks to its efficient memory writing on GPUs. For a tensor with 512MB, the encoding time of DRAGONN and DGC is 13.2ms and 35.4ms, respectively. As a comparison, the model size of XML with Wiki10-31K dataset is 507MB and its single-GPU iteration time is 25 ms. Therefore, DRAGONN reduces the major overhead in the DDT of XML. In addition, we observe that the encoding time almost keeps constant with the compression ratio ranging from 0.0001 to 0.01, which is consistent with the number of operations listed in Table 1.

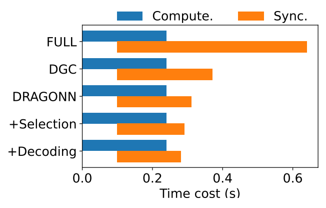
Figure 5(b) compares the performance of dense decoding and sparse decoding with the tensor size as 64MB and the compression ratio as 0.001. The decoding time of dense decoding linearly increases with the number of GPUs because of the conversion from sparse tensors into dense formats. In contrast, the decoding time of sparse decoding keeps



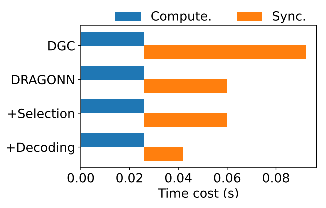
(a) ResNet50



(b) ViT

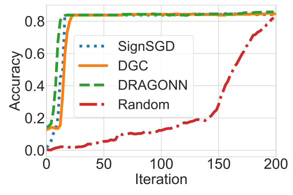


(c) MLP-Mixer

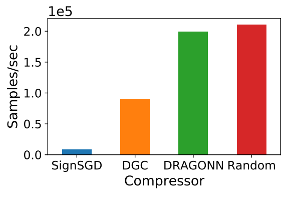


(d) XML

Figure 6. Impacts of the three components (hashing-based compressor, efficient-aware tensor selection and sparse decoding) on the per-iteration time costs of the computation (Compute.) and synchronization (Sync.).

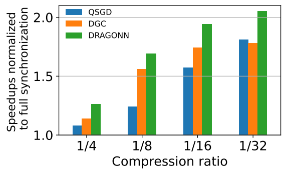


(a)

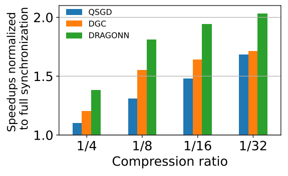


(b)

Figure 7. For DDT of XML Wiki10-31K model over 16GPUs, we plot (a) accuracy versus number of iteration, (b) training speed of SignSGD, DGC, Random Sampling (Random) and DRAGONN. We set the compression ratio of SignSGD as $\frac{1}{32}$ (minimum permitted ratio). We set compression ratio as 10^{-3} for other methods.



(a) MLP-Mixer



(b) ViT

Figure 8. Training speedups versus the compression ratios.

constant thanks to the batching mechanism. Note that the speedups of sparse decoding is over $K\times$, where K is the number of GPUs, because it also saves the computational overhead of aggregations.

We then evaluate the individual performance gains of the three components in end-to-end training with 16 GPUs. The computation latency is the time for forward and backward propagation, and it keeps constants in DDT (Zhang et al., 2020; Li et al., 2020). The synchronization latency combines the communication and compression time. It can overlap with computation in the training of ResNet50, ViT, and MLP-Mixer, as shown in Figure 6(b) and 6(c). The speedups of DRAGONN-only (i.e., only applying the hashing-based compressor) over DGC are $1.33\times$, $1.21\times$, and $1.25\times$ for ResNet50, ViT, and MLP-Mixer in terms of the synchronization latency. The two system optimization techniques can further increase the speedups to $1.92\times$, $1.35\times$, and $1.39\times$, respectively. Because XML has only one layer, there is no overlap between its computation and synchronization. Figure 6(d) shows that the speedup of DRAGONN-only for its synchronization latency is $1.91\times$. The sparse decoding mechanism can further increase the speedup to $4.33\times$.

5.3. More Comparisons

In this section, we compare DRAGONN with two other representative compression algorithms, i.e., GS with random sampling (Random) (Stich et al., 2018) and SignSGD (Karimireddy et al., 2019), which is a 1-bit quantization algorithm. The evaluated model is XML on Wiki10-31K dataset and the number of GPUs is 16. Note that the compression ratio is $\frac{1}{32}$ for SignSGD and 10^{-3} for others. Figure 7(a) shows that DRAGONN outperforms baselines with less iterations to converge, even when SignSGD keeps $31.25\times$ gradient values than DRAGONN. Figure 7(b) displays the training speed of four methods. DRAGONN outperforms DGC and SignSGD with faster speed. Although random sampling outperforms DRAGONN in the training speed, it requires $15.1\times$ iterations to converge. Overall, DRAGONN achieves $14.1\times$ speedup over random sampling. We also compare the speedups of DRAGONN, DGC, and QSGD (Alistarh et al., 2017) in different compression ratio for ViT and MLP-Mixer in Figure 8.

6. Related Work

Besides DGC, many GS algorithms are proposed to address the communication bottlenecks by reducing the traffic volume. The TopK algorithm (Aji & Heafield, 2017) is a well-known GS paradigm to reduce the traffic volume for distributed training. Unfortunately, its exorbitant operation overhead outruns the saved communication time and can harm the practical performance (Xu et al., 2021; Shi et al., 2021; Agarwal et al., 2021). RandomK (Stich et al., 2018) and its extension (Barnes et al., 2020) randomly selects a subset of gradient for synchronization, but they may downgrade the model's performance in convergence rate. Among the method proposed above, the compression time still performs as the major efficiency bottleneck. MSTopk (Shi et al.,

2021) also notices the compression overhead in DGC, but the focus is the threshold search process and it compresses all tensors for communication.

Because the aggregation operations of sparse tensors are not associative (Xu et al., 2021; Agarwal et al., 2021), Allreduce (Thakur et al., 2005) cannot be used for the gradient synchronization of compression-enabled DDT. Several algorithms, such as Ok-Topk (Li & Hoefler, 2022), SparCML (Renggli et al., 2019), and gTopk (Shi et al., 2019), are recently proposed to optimize communication for the aggregation of sparse tensors after compression operation. However, DRAGONN focuses on directly minimizing the operation cost in extracting the topk elements before communication. DRAGONN and these algorithms solve different problems and thus, are orthogonal and complementary.

Other than gradient sparsification, quantization is another popular types of gradient compression algorithms. It decreases the precision of gradients to reduce the traffic volume. Gradients in FP32 can be mapped to fewer bits, such as 8 bits (Dettmers, 2015; Alistarh et al., 2017), 2 bits (Wen et al., 2017), and even 1 bit (Seide et al., 2014; Karimireddy et al., 2019). The compression rate of quantization is at most $32\times$, much lower than that of GS. These works focus on the design of compression algorithms, but they do not consider the system-level optimization for the overall performance.

7. Conclusion

In this work, we propose DRAGONN: a randomized gradient sparsification (GS) algorithm for data-parallel distributed training (DDT) of neural networks. We identify the overhead of current GS approaches and propose a hashing-based randomized algorithm to tackle this bottleneck. We show that, with our compression techniques, DRAGONN achieves up to $3.52\times$ speedup over the current state-of-the-art GS algorithm. Moreover, the promising scalability of DRAGONN indicates its strength in DDT with increasing number of workers. We hope DRAGONN would open the door for more randomized algorithm in DDT.

8. Acknowledgement

Zhuang Wang, Xinyu Crystal Wu and T. S. Eugene Ng are supported by the National Science Foundation CNS-1718980, CNS-1801884, and CNS-1815525. Zhaozhuo Xu and Anshumali Shrivastava are supported by the National Science Foundation IIS-1652131, BIGDATA-1838177, AFOSR-YIP FA9550-18-1-0152, the ONR DURIP Grant, the ONR BRC grant on Randomized Numerical Linear Algebra and the Ken Kennedy Institute BP fellowship. We would like to thank Ben Coleman and Zhenwei Dai for helpful discussions, and the anonymous reviewers for their valuable feedback.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016.

Agarwal, S., Wang, H., Venkataraman, S., and Papailiopoulos, D. On the utility of gradient compression in distributed training systems. *arXiv preprint arXiv:2103.00543*, 2021.

Aji, A. F. and Heafield, K. Sparse communication for distributed gradient descent. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Bai, Y., Li, C., Zhou, Q., Yi, J., Gong, P., Yan, F., Chen, R., and Xu, Y. Gradient compression supercharged high-performance data parallel dnn training. In *The 28th ACM Symposium on Operating Systems Principles (SOSP 2021)*, 2021.

Barnes, L. P., Inan, H. A., Isik, B., and Özgür, A. rtop-k: A statistical estimation approach to distributed sgd. *IEEE Journal on Selected Areas in Information Theory*, 1(3): 897–907, 2020.

Basu, D., Data, D., Karakus, C., and Diggavi, S. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems*, 32, 2019.

Ben-Nun, T. and Hoefler, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

Bhatia, K., Dahiya, K., Jain, H., Kar, P., Mittal, A., Prabhu, Y., and Varma, M. The extreme classification repository: Multi-label datasets and code, 2016. URL `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Blelloch, G. E. Prefix sums and their applications. In *Sythesis of parallel algorithms*, pp. 35—60. Morgan Kaufmann Publishers Inc., 1990.

Carter, J. L. and Wegman, M. N. Universal classes of hash functions. *Journal of computer and system sciences*, 18 (2):143–154, 1979.

Chen, B., Liu, Z., Peng, B., Xu, Z., Li, J. L., Dao, T., Song, Z., Shrivastava, A., and Re, C. Mongoose: A learnable lsh framework for efficient neural network training. In *International Conference on Learning Representations*, 2020.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 578–594, 2018.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Desai, A., Xu, Z., Gupta, M., Chandran, A., Vial-Aussavy, A., and Shrivastava, A. Raw nav-merge seismic data to subsurface properties with mlp based multi-modal information unscrambler. *Advances in Neural Information Processing Systems*, 34, 2021.

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

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

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

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

Google. TensorFlow where(). https://www.tensorflow.org/api_docs/python/tf/where, 2022.

Harris, M., Sengupta, S., and Owens, J. D. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39):851–876, 2007.

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

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., and Guo, C. A unified architecture for accelerating distributed {DNN} training in heterogeneous gpu/cpu clusters. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 463–479, 2020.

Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Li, P., Li, X., and Zhang, C. H. Re-randomized densification for one permutation hashing and bin-wise consistent weighted sampling. *Advances in Neural Information Processing Systems*, 32, 2019.

Li, S. and Hoefler, T. Near-optimal sparse allreduce for distributed deep learning. *arXiv preprint arXiv:2201.07598*, 2022.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al. Pytorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13 (12):3005–3018, 2020.

Li, X. and Li, P. Rejection sampling for weighted jaccard similarity revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4197–4205, 2021a.

Li, X. and Li, P. C-minhash: Rigorously reducing k permutations to two. *arXiv preprint arXiv:2109.03337*, 2021b.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. *The International Conference on Learning Representations (ICLR)*, 2017.

Liu, H., Dai, Z., So, D. R., and Le, Q. V. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021.

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

Meta. PyTorch nonzero(). https://pytorch.org/docs/stable/generated/torch.nonzero.html, 2022.

Mitzenmacher, M. and Upfal, E. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

Naumov, M., Mudigere, D., Shi, H. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., Chen, W., Rao, V., Jia, B., Xiong, L., and Smelyanskiy, M. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019. URL https://arxiv.org/abs/1906.00091.

Nvidia. Nccl lantecy. https://developer.nvidia.com/blog/scaling-deep-learning-training-nccl/, 2018.

NVIDIA. A Timeline of Innovation for NVIDIA. https://www.nvidia.com/en-us/about-nvidia/corporate-timeline/, 2021.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

Patarasuk, P. and Yuan, X. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Renggli, C., Ashkboos, S., Aghagolzadeh, M., Alistarh, D., and Hoefler, T. Sparcml: High-performance sparse communication for machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2019.

Ridnik, T., Ben-Baruch, E., Noy, A., and Zelnik-Manor, L. Imagenet-21k pretraining for the masses, 2021.

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

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

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

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20:1–49, 2019.

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

Shi, S., Zhou, X., Song, S., Wang, X., Zhu, Z., Huang, X., Jiang, X., Zhou, F., Guo, Z., Xie, L., et al. Towards scalable distributed training of deep learning on public cloud clusters. *Proceedings of Machine Learning and Systems*, 3, 2021.

Shrivastava, A. and Li, P. Densifying one permutation hashing via rotation for fast near neighbor search. In *International Conference on Machine Learning*, pp. 557–565. PMLR, 2014.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. Sparsified sgd with memory. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 4452–4463, 2018.

Strom, N. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Sun, P., Feng, W., Han, R., Yan, S., and Wen, Y. Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes. *arXiv preprint arXiv:1902.06855*, 2019.

Thakur, R., Rabenseifner, R., and Gropp, W. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 19(1), 2005.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Keysers, D., Uszkoreit, J., Lucic, M., et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

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

Wang, Z., Wu, X., and Ng, T. Mergecomp: A compression scheduler for scalable communication-efficient distributed training. *arXiv preprint arXiv:2103.15195*, 2021.

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pp. 1509–1519, 2017.

Xu, H., Ho, C.-Y., Abdelmoniem, A. M., Dutta, A., Bergou, E. H., Karatsenidis, K., Canini, M., and Kalnis, P. Grace: A compressed communication framework for distributed machine learning. In *Proc. of 41st IEEE Int. Conf. Distributed Computing Systems (ICDCS)*, 2021.

Zhang, Z., Chang, C., Lin, H., Wang, Y., Arora, R., and Jin, X. Is network the bottleneck of distributed training? In *Proceedings of the Workshop on Network Meets AI and ML*, NetAI '20, pp. 8–13. Association for Computing Machinery, 2020. ISBN 9781450380430.

Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., et al. Ansor: Generating high-performance tensor programs for deep learning. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 863–879, 2020.

Zhou, X., Urata, R., and Liu, H. Beyond 1 tb/s intra-data center interconnect technology: Im-dd or coherent? *Journal of Lightwave Technology*, 38(2):475–484, 2020.

Zinkevich, M., Weimer, M., Smola, A. J., and Li, L. Parallelized stochastic gradient descent. In *NIPS*, volume 4, pp. 4. Citeseer, 2010.

Appendix

A. Theoretical Analaysis

In this section, we perform the theoretical analysis on DRAGONN for gradient sparsification. We start with providing the settings. Next, we bound the compression error of DRAGONN. Next, we present the generalization error bound for DRAGONN. Finally, we bound the memory coverage of DRAGONN and provide some case study.

A.1. Settings

Notations: We use $\Pr[]$ and $\mathbb{E}[]$ for probability and expectation. We use $\max(S)$ and $\min(S)$ to denote the maximum and minimum value in the set $S \in \mathbb{N}$. For a vector $x \in \mathbb{R}^d$, we use $\|x\|_2 := (\sum_{i=1}^d x_i^2)^{1/2}$ to denote its ℓ_2 norm.

In this paper, we study the function $g(\theta; x, y)$ parameterized by $\theta \in \mathbb{R}^d$ with respect to data-label pair (x, y). Let D denotes the dataset. In the distributed setting with K nodes, we generate $D_1, D_2, \ldots, D_K$ so that each node trains on its own subset $D_i \subset D$. We would like the ℓ_2 norm of the gradient of $g(\theta; x, y)$ with respect to θ to be bounded as:

Definition A.1 (Bounded gradient norm)**.** *Given a loss function $g(\theta; x, y)$ with respect to a dataset D, we say the gradient $\nabla_\theta g(\theta; x, y)$ is G bounded if for any $x \in D$,*

$$\|\nabla_\theta g(\theta; x, y)\|_2^2 \leq G^2,$$

where $G > 0$ is the upper bound of the gradient norm.

In our work, we would like to optimize the empirical risk $f(\theta) = \frac{1}{K}\sum_{i=1}^K \mathbb{E}_{(x,y)\sim D_i}[g(\theta; x, y)]$. Let $f_i(\theta) = \mathbb{E}_{(x,y)\sim D_i}[g(\theta; x, y)]$. We would like the function $f_i(\theta)$ to be L-smooth defined as below

Definition A.2 (Smoothness)**.** *We say the loss function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if for any $a, b \in \mathbb{R}^d$,*

$$f(b) \leq f(a) + \langle \nabla f(a), b - a\rangle + \frac{L}{2}\|b - a\|_2^2.$$

Note that these settings are standard in related literature (Bhatia et al., 2016; Barnes et al., 2020).

A.2. Compression Error

In this section, we upper bound the compression error of the hashing-based compressor shown in Figure 2. We start with defining the compression error.

Definition A.3 (Compression error)**.** *For a weight $\theta \in \mathbb{R}^d$, let $\phi(\theta) : \mathbb{R}^d \to \mathbb{R}^d$ denotes the operation that satisfy θ. We define the compression error as $\|\theta = \phi(\theta)\|_2^2$.*

The error results from the selection of gradients and the hash collisions in the memory writing of the gradient indices. In this work, we use the propriety of universal hashing function (Carter & Wegman, 1979) defined as below:

Definition A.4 (Universal hashing (Carter & Wegman, 1979))**.** *Let $m \in \mathbb{N}_+$. A family of function $\mathcal{H}$ is a universal hashing function if it follows that for any $h : \mathbb{N} \to [m]$ from family $\mathcal{H}$, for any $x, y \in \mathbb{N}$*

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}.$$

Universal hashing is the foundation of most hashing algorithms (Shrivastava & Li, 2014; Li et al., 2019; Li & Li, 2021a;b). Next, we present several supporting lemmas to upper bound the compression error. We first provide the probability of a successful memory writing for a gradient.

Lemma A.5. *Given a universal hashing function $h : \mathbb{N} \to [m]$ from family $\mathcal{H}$, for a set of values $S = \{s_1, s_2, \cdots, s_n\} \subset \mathbb{R}$, we show that for $i \in [n]$,*

$$\mathbb{E}\left[1(i = \max(\{j \in [n] | h(j) = h(i)\}))\right] = (\frac{m-1}{m})^{n-i}$$

Proof. We start with showing that

$$\begin{aligned}\Pr[i = \max(\{j|h(j) = h(i)\})] = &\ \Pr[\forall j > i, h(j) \neq h(i)] \\ = &\ (\frac{m-1}{m})^{n-i}\end{aligned}$$

Therefore, following the definition of expectation, we have

$$\mathbb{E}\left[1(i = \max(\{j|h(j) = h(i)\}))\right] = (\frac{m-1}{m})^{n-i}$$

□

Next, we show the upper bound of the approximation errors introduced by hashing.

Lemma A.6. *Let* $m, n \in \mathbb{N}$ *and* $n > m$. *Let* $S = \{s_1, s_2, \cdots, s_n\} \subset \mathbb{R}$ *denotes an* n*-point set of values. Let* $\tau = \max(S)/\min(S)$. *we have*

$$\sum_{i=1}^{n}\left(1 - \left(\frac{m-1}{m}\right)^{n-i}\right) s_i^2 \leq (1-\gamma)\sum_{i=1}^{n} s_i^2$$

where $\gamma = \frac{m}{n\tau^2}(1 - (\frac{m-1}{m})^n)$.

Proof. Let $s_{\max} = \max(S)$ and $s_{\min} = \min(S)$. We show that

$$\begin{aligned}&(1-\gamma)\sum_{i=1}^{n} s_i^2 - \sum_{i=1}^{n}(1 - (\frac{m-1}{m})^{n-i})s_i^2 \\ &= \sum_{i=1}^{n} s_i^2\left(1 - \gamma - 1 + \left(\frac{m-1}{m}\right)^{n-i}\right) \\ &= \sum_{i=1}^{n} s_i^2\left(\left(\frac{m-1}{m}\right)^{n-i} - \gamma\right) \\ &\geq s_{\min}^2(m(1 - (\frac{m-1}{m})^n) - s_{\max}^2 n\gamma \\ &\geq s_{\min}^2(m(1 - (\frac{m-1}{m})^n) - \tau^2 n\gamma) = 0\end{aligned}$$

where the first and second steps are reorganzations, the third step follows by the definition of $s_{\max}$ and $s_{\min}$, the forth step follows from the definition of τ, the last step follows from the definition of γ. □

Lemma A.7. *Let* $S = \{s_1, s_2, \cdots, s_n\} \subset \mathbb{R}$ *denotes an* n*-point set of values. Let* $\tau = \max(S)/\min(S)$. *Given an universal hashing function* $h : \mathbb{N} \rightarrow [m]$ *from family* $\mathcal{H}$, *we show that*

$$\begin{aligned}&\mathbb{E}\left[\sum_{i=1}^{n}\left(1 - 1\left(i = \max(\{j|h(j) = h(i)\})\right)\right)^2 s_i^2\right] \\ \leq &\ (1-\gamma)\sum_{i=1}^{n} s_i^2\end{aligned}$$

where $\gamma = \frac{m}{n\tau^2}(1 - (\frac{m-1}{m})^n)$.

Proof.

$$
\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{n} (1 - 1\,(i = \max\{j|h(j) = h(i)\}))^2\, s_i^2\right] \\
&= \sum_{i=1}^{n} s_i^2 \mathbb{E}\left[(1 - 1\,(i = \max\{j|h(j) = h(i)\}))^2\right] \\
&= \sum_{i=1}^{n} s_i^2 - 2\sum_{i=1}^{n} s_i^2 \mathbb{E}\left[1(i = \max\{j|h(j) = h(i)\})\right] \\
&\quad + \sum_{i=1}^{n} s_i^2 \mathbb{E}\left[1(i = \max\{j|h(j) = h(i)\})^2\right] \\
&= \sum_{i=1}^{n} s_i^2 - \sum_{i=1}^{n} s_i^2 \mathbb{E}\left[1(i = \max\{j|h(j) = h(i)\})\right] \\
&= \sum_{i=1}^{n} s_i^2 \left(1 - \left(\frac{m-1}{m}\right)^{n-i}\right) \\
&\le (1-\gamma)\sum_{i=1}^{n} s_i^2
\end{aligned}
$$

where the first and second steps are reorganzations, the third step follows from $\mathbb{E}\left[1(i = \max\{j|h(j) = h(i)\})^2\right] = \mathbb{E}\left[1(i = \max\{j|h(j) = h(i)\})\right]$, the forth step follows from Lemma A.5, the fifth step follows form Lemma A.6. □

Finally, we provide the upper bound of the compression error for DRAGONN.

Lemma A.8. *For a weight $\theta \in \mathbb{R}^d$, let $\phi(\theta)$ denotes an sparsification operation (see Definition A.3) that takes top n values of θ and keep the value that has been successfully hashed it into the memory via universal hashing function, we show that*

$$
E\|\theta - \phi(\theta)\| \le (1 - \frac{m(1 - (\frac{m-1}{m})^n)}{d\tau^2})\|\theta\|_2
$$

Proof. Let $\{\theta_1, \cdots, \theta_n\}$ denotes the top n values. Let $\{\theta_{n+1}, \cdots, \theta_d\}$ the rest of the values. We show that

$$
\begin{aligned}
E\|\theta - \phi(\theta)\| &\le (1 - \frac{m(1 - (\frac{m-1}{m})^{n\tau^2})}{n})\sum_{j=1}^{n}\theta_j^2 + \sum_{j=n+1}^{d}\theta_j^2 \\
&\le \sum_{j=1}^{d}\theta_j^2 - \frac{m(1 - (\frac{m-1}{m})^n)}{n\tau^2}\frac{n}{d}\sum_{j=1}^{n}\theta_j^2 \\
&= (1 - \frac{m(1 - (\frac{m-1}{m})^n)}{d\tau^2})\|\theta\|_2
\end{aligned}
$$

where the first step follows from Lemma A.7, the second step follows from $\theta_i \ge \theta_j$ if $i \le n$ and $j > n$. the last step is an reorganization. □

A.2.1. GENERALIZATION ERROR

In this section, we showcase the upper bound of the generalization error if we follow the setting in (Basu et al., 2019; Barnes et al., 2020). The upper bound is shown in Theorem A.9.

Theorem A.9. *Let $f(\theta) = \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}_{(x,y)\sim D_i}[g(\theta; x, y)]$ denotes a L-smooth (Definition A.2) empirical risk function with $\nabla_\theta g(\theta; x, y)$ being G bounded (Definition A.1). Let the learning rate to be $\eta = \frac{1}{2L}$. Starting from a weight θ_0, after T step*

Algorithm 3 Existing approximate TopK GS

```
Input: gradient $\mathcal{G}$, tensor size $d$, threshold $t$
Output: $\mathcal{G}_c$, $I$
$mask \leftarrow |\mathcal{G}| > t$
$d \leftarrow \mathsf{len}(mask)$
$I \leftarrow \emptyset$
$s \leftarrow 0$
$s_{\mathsf{prev}} \leftarrow 0$
for $i = 0$ to $d - 1$ do
  $s \leftarrow s + mask[i]$
  if $s > s_{\mathsf{prev}}$ then
    $I \leftarrow I \cup i$
    $s_{\mathsf{prev}} \leftarrow s$
  end if
end for
$\mathcal{G}_c = \mathcal{G}[I]$
```

we would obtain a θ_T through stochastic gradient descent with DRAGONN compressor (Algorithm 1) such that

$$\mathbb{E}[\|\nabla f(\theta_T)\|_2^2] \leq \frac{8L\mathbb{E}[f(\theta_0) - f^*]}{T} + \frac{2G^2}{Kd} + (\frac{8}{\epsilon^2} - 6)G^2$$

where $\epsilon = \frac{m(1-(\frac{m-1}{m})^n)}{d\tau^2}$.

The proof of the theorem is done by applying the compression error $\epsilon = \frac{m(1-(\frac{m-1}{m})^n)}{d\tau^2}$ in Lemma A.8 in to the setting of (Basu et al., 2019).

A.2.2. Memory Coverage

In this section, we discuss the memory coverage of DRAGONN. Due to the collision of the universal hashing function. The exists empty memory locations if we write n gradient values into m locations. We relate this problem as a Balls-and-Bins (Mitzenmacher & Upfal, 2017) problem: given n balls, we want to toss them into m bins via universal hashing. We start with indexing the balls with $\{1, \cdots, n\}$ and then hash the index for each ball via a universal hashing function $h : \mathbb{N} \to [m]$. Therefore, the probability of one bin being empty $(1 - \frac{1}{m})^n$. Moreover, let X be the number of empty bins. Let x be the ratio of empty bins in this m bins, for any $0 < \epsilon < 1$, we have.

Lemma A.10 (Bound on the ratio of empty bins (Mitzenmacher & Upfal, 2017))**.** *Let x be the ratio of empty bins in m bins. Let $p = (1 - \frac{1}{m})^n$. We bound the coverage x as,*

$$P(|x - p| \geq \epsilon) \leq 2e\frac{e^{-m\epsilon^2/3p}}{\sqrt{m}}$$

We provide several examples to illustrate this bound. If we set $m = 1024$, $n = 1024$ and $\epsilon = 0.1$, with probability at least 0.95, we have $0.36 \leq x \leq 0.38$. If we set $m = 512$, $n = 1024$ and $\epsilon = 0.1$, with probability at least 0.8 we have $0.13 \leq x \leq 0.14$.

B. Baseline Algorithm

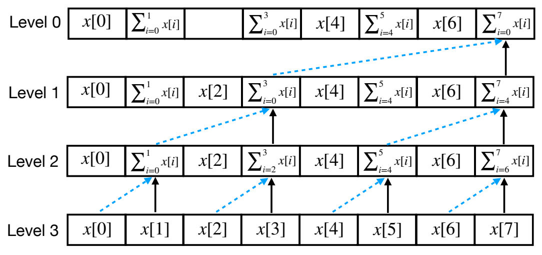
We present existing approximate TopK GS in Algorithm 3.

C. Parallel Prefix Sum

This section describes the definition of prefix sum and the parallel prefix sum algorithm with CUDA (Harris et al., 2007). Following (Blelloch, 1990), we define the prefix-sum operation as:

Definition C.1. *The prefix-sum operation performs as follow: given a binary associative operator $\oplus$, for an n entries array*

$$[a_0, a_1, \ldots, a_{n-1}],$$


(a) The up-sweep phase

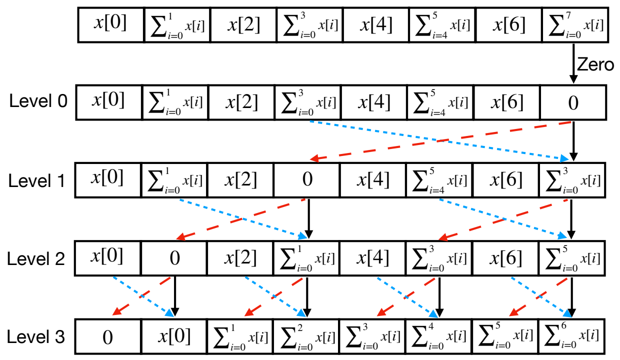

(b) The down-sweep phase

Figure 9. An illustration of the up-sweep and down-sweep phases of parallel prefix sum algorithm.

Algorithm 4 The up-sweep phase

```
**Input:** array $x$, number of elements $d$
**for** $i = 0$ **to** $\log_2 d - 1$ **do**
  **for** $r = 0$ **to** $d - 1$ **by** $2^{i+1}$ **in parallel do**
    $x[r + 2^{i+1} - 1] \leftarrow x[r + 2^i - 1] + x[r + 2^{i+1} - 1]$
  **end for**
**end for**
```

Algorithm 5 The down-sweep phase

```
**Input:** array $x$, number of elements $d$
$x[d - 1] \leftarrow 0$
**for** $i = \log_2 d$ **to** $0$ **by** $-1$ **do**
  **for** $r = 0$ **to** $d - 1$ **by** $2^{i+1}$ **in parallel do**
    $x[r + 2^{i+1} - 1] \leftarrow x[r + 2^i - 1] + x[r + 2^{i+1} - 1]$
  **end for**
**end for**
```

we return the array

$$[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots a_{n-1})].$$

We consider $\oplus$ as addition in this paper.

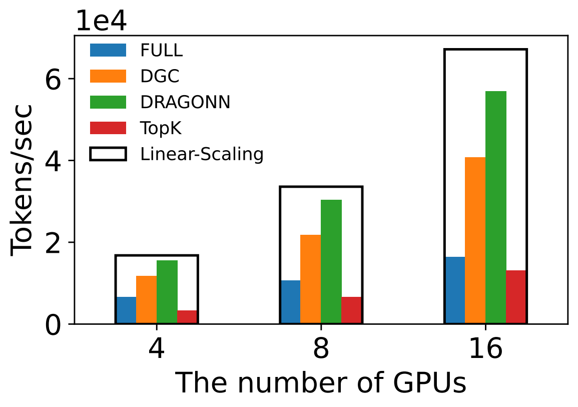
Given an input, parallel prefix sum conceptually constructs a balanced binary tree. It sweeps the data to and from the root of tree to compute the prefix sum. Suppose a binary tree has d leaves, it has $\log d$ levels and the number of nodes in each level $i \in [0, \log d)$ is 2^i. Parallel prefix sum has two phases: *up-sweep phase* and *down-sweep phase*. In the up-sweep phase, as shown in Figure 9(a), the algorithm traverses the tree from leaves to root to calculate partial sums at internal nodes. In the down-sweep phase, as shown in Figure 9(b), it traverse back up the tree from the root. The partial sums calculated in the up-sweep phase are used to build the scan in place on the array. The algorithms of the two phases are illustrated in Algorithm 4 and Algorithm 5

After the prefix-sum operation for the mask, DGC needs to compare each element with its neighbor on its left-hand side to extract the indices. If it is greater than its neighbor, DGC writes the index of this element into the position with the offset of its value.

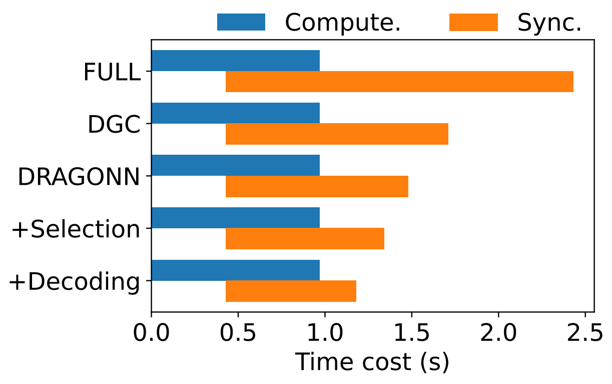
DGC needs d comparison operations to get the mask, $(d-1)$ add operations for up-sweep phase, $(d-1)$ add operations and $(d-1)$ swap operations for down-sweep phase, and d comparison operations after prefix-sum operation. In total, DGC uses $2d$ comparison operations, $2(d-1)$ add operations, and $(d-1)$ swap operations to extract the indices of top-k gradients.

D. Experiments over BERT.

We fine-tune BERT-base for the question-answering task. The dataset is SQuAD1.1, the batch size is 1024 tokens, and the compression ratio is 0.1%.



(a) Training speed



(b) Impacts breakdown

Figure 10. Experiments on BERT-base.

Figure 10(a) shows the training speed with different numbers of GPUs via different compressors. With 16 GPUs, DRAGONN performs 1.34x speedup over DGC and 3.5x speedup over full synchronization. Moreover, Figure 10(b) breaks down the impacts of the three techniques proposed in DRAGONN.