Beyond Throughput and Compression Ratios: Towards High End-to-end Utility of Gradient Compression

Wenchen Han
University College London

Shay Vargaftik
VMware Research

Michael Mitzenmacher
Harvard University

Brad Karp
University College London

Ran Ben Basat
University College London

ABSTRACT

Gradient aggregation has long been identified as a major bottleneck in today's large-scale distributed machine learning training systems. One promising solution to mitigate such bottlenecks is gradient compression, directly reducing communicated gradient data volume. However, in practice, many gradient compression schemes do not achieve acceleration of the training process while also preserving accuracy.

In this work, we identify common issues in previous gradient compression systems and evaluation methodologies. These include excessive computational overheads; incompatibility with all-reduce; and insufficient evaluation methods, such as not using an end-to-end metric or using a 32-bit baseline instead of the stronger 16-bit baseline. We revisit common compression approaches (sparsification, quantization, and low-rank decomposition) and demonstrate how considering the above issues can lead to minor but strategic design changes, resulting in notably better performance. Our goal is to raise awareness of the need for design and evaluation standards that naturally translate to the end-to-end utility of gradient compression.

CCS CONCEPTS

• **Computer systems organization** → *Distributed architectures*; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Gradient compression, Collective communication, All-reduce.

HOTNETS '24, November 18–19, 2024, Irvine, CA, USA

ACM ISBN 979-8-4007-1272-2/24/11
https://doi.org/10.1145/3696348.3696857

ACM Reference Format:
Wenchen Han, Shay Vargaftik, Michael Mitzenmacher, Brad Karp, and Ran Ben Basat. 2024. Beyond Throughput and Compression Ratios: Towards High End-to-end Utility of Gradient Compression. In *The 23rd ACM Workshop on Hot Topics in Networks (HOTNETS '24), November 18–19, 2024, Irvine, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3696348.3696857

1 INTRODUCTION

Distributed Data-Parallel (DDP) training [21] is the de-facto paradigm for large-scale distributed machine learning training systems. A key obstacle to efficient DDP training is the large communication volume of aggregating and synchronizing the gradients [43, 53, 59, 60]. Further, the training hardware's processing speed advances faster than the network bandwidth [7], exacerbating this issue.

One promising direction to alleviate the communication bottleneck is to apply a gradient compression scheme [11, 14, 16, 18, 20, 23, 30, 32, 34, 36, 51, 57, 58, 60, 62, 63]. Gradient compression aims to reduce the communicated data volume [59]. Most schemes are lossy, meaning they introduce some compression error, which the schemes try to minimize. Nevertheless, as pointed out by prior studies [11, 62], compression schemes often end up yielding a limited speedup in practice or compromise the model's accuracy.

We seek to uncover the root causes that lead to degraded end-to-end utility in practice. We classify our findings into two categories, design and evaluation, and exemplify how to address these issues via a case study (Section 3) that covers three main gradient compression types: sparsification (TopK [12, 51]), quantization (THC [34]), and low-rank decomposition (PowerSGD [57]).

Design. One common issue is the compression's computational overhead [11, 62, 64]. Our study finds that some components of compression incur superlinear computational complexity [24, 25, 34, 57]; others have GPU-inefficient memory access patterns [24, 25, 38]. Another challenge is that many compression schemes are incompatible with the all-reduce collective [9, 41], which is inherently more scalable than the all-gather collective that generates higher traffic overhead, or the parameter server aggregation [33] that has one-to-many and many-to-one communication patterns [43, 46, 56].

We elaborate on these particular issues in our case study (Section 3), and based on our consideration, propose techniques such as Chunking, Partial Rotation, and Saturation to improve performance.

Evaluation. Gradient compression studies (e.g., [11, 14, 15, 23, 30, 32, 34, 36, 60, 62]) often choose the compression ratio (the amount of reduced communication volume) and throughput as their design objectives and evaluation metrics, and compare with a full precision (FP32) baseline. We argue that such an evaluation is insufficient for the following reasons. First, neither the throughput nor the compression ratio reflect the accuracy degradation. In an end-to-end sense, gradient compression aims at optimizing the time it takes to reach a target accuracy, *i.e.*, the time to accuracy (TTA). An excessively aggressive scheme to cut down the communication overhead may improve the throughput but often results in degraded TTA due to the high compression error that dominates the convergence speed. Second, half-precision (FP16) [1] is found to be a stronger baseline. This is because it requires half the number of bits and is widely supported in ML hardware, which can commonly make more FP16 ops per second compared to FP32 [5, 8, 26]. Accordingly, FP16 compression and aggregation are found to be [27] more performant in terms of TTA than FP32. The evaluation is therefore convincing only if the gradient compression scheme outperforms the higher FP16 bar. We refer to the TTA improvement over this FP16 baseline as a method's *utility*. We demonstrate the usefulness of the proposed evaluation methods using the TTA metric and FP16 baseline in our case study.

In summary, we make the following contributions:

1) We identify common design issues of gradient compression systems that restrain their training speedup in practice or compromise the model's accuracy.

2) We identify common evaluation insufficiencies that may lead to suboptimal design choices that do not translate to improved end-to-end utility in practice.

3) We exemplify these issues in a case study covering three main types of gradient compression schemes and propose optimization techniques that address these issues resulting in better utility.

2 GRADIENT COMPRESSION CHALLENGES

In this section, we overview common fallacies and issues in the design and evaluation of modern gradient compression schemes. In particular, we focus on the design issues (that result in degraded end-to-end performance) associated with high computational overhead and incompatibility with all-reduce, the de facto standard for DDP. We also consider evaluation practices that insufficiently model the end-to-end utility of a gradient compression system, including the usage of metrics that do not capture end-to-end performance, and inappropriate or weak baselines. We pick several state-of-the-art system papers of gradient compression [11, 14, 23, 30, 32, 34, 60, 62] for analysis, as listed in Table 1.

2.1 Design Issues

Computational overhead. As shown in previous studies [11, 62, 64], gradient compression often underperforms an uncompressed baseline in terms of its training throughput. Computational overhead for compression plays a key role here. In this work, we identify components that create bottlenecks through excessive computation or using the GPU's global memory [38] with inefficient memory access patterns.

Incompatibility with all-reduce. There are two common types of gradient collection methods: collection at a centralized parameter server (PS) [33], and decentralized collective operations such as all-gather and all-reduce [9, 41]. All-reduce (including ring all-reduce [2] and tree all-reduce [42]) is inherently more scalable than all-gather [9, 41] and PS aggregation [33], since many-to-one communications [46, 56] incur temporal congestion, and RDMA NICs face dropped performance maintaining too many connections [61]. Recently, [28] explored the PS co-located mode that reduces the temporal load on any specific worker, but still suffers from the many-to-one and one-to-many communications.

Unfortunately, previous gradient compression algorithms are often not compatible with all-reduce collectives [11]. In all-reduce, unlike in the PS architecture, workers on an intermediate hop receive partially aggregated gradients, add their own, and then send the updated result to the next hop. The difficulty lies in avoiding decompression and recompression at each worker, as this would lead to significant computational overheads and accumulation of compression-induced errors, which could negatively impact the training process.

2.2 Evaluation Issues

The choice of an end-to-end metric. Prior works [11, 14, 23, 30, 32, 34, 60, 62] primarily focus on optimizing the *training throughput* metric. Admittedly, training throughput can be a cheap metric to evaluate on, as it does not require a full training process. Also, training throughput is a suitable metric for lossless compression, where compression error need not be taken into account. However, with lossy compression, training throughput is not an end-to-end metric, as it does not consider the impact of compression error on the model's accuracy [1]. As shown in Table 1, several prior gradient compression systems do not take into account compression error in their design, do not include end-to-end performance evaluation, or do not achieve acceleration compared to the baselines in terms of time-to-accuracy.

[1]Here accuracy refers to the goal metric of the trained model, such as classification accuracy in recognition tasks and perplexity (the model's confidence in predicting the correct next token) for language modeling.

	[11]	[14]	[23]	[30]	[32]	[34]	[60]	[62]
Comparing with the stronger FP16 baseline	✗	✗	✗	✗	✗	✗	✗	✗
Considering compression error for system design	N/A	✗	✓	N/A	✓	✓	✗	✗
Evaluation on end-to-end performance (in how many tasks)	0/3	2/8	1/6	3/4	4/4	3/7	4/4	3/3
Higher throughput results in better time to accuracy	N/A	✓	✓	✓	✗	✓	✓	✗
All-reduce compatibility for new compression algorithms	N/A	N/A	✗	✓	✓	✗	N/A	✗

Table 1: Assessment of prior gradient compression systems. Here, 'N/A' means that the criterion is not applicable.

In an end-to-end sense, the utility of a gradient compression scheme is exactly the training time saved to train a model. Accordingly, we propose that the time to accuracy (TTA) should be the main end-to-end metric. Critically, TTA is a *2-dimensional* metric in which for each accuracy target, an algorithm has a measured training time required to meet that target. One challenge is that each compression scheme is then represented by a *curve*, not a single data point, and the curves can intersect, so that which scheme is better can depend on the setting. We argue here that research papers, in particular, should present *TTA curves* when comparing compression schemes, as otherwise, they are leaving out the key information for understanding comparative performance.

While TTA curves still leave room for interpretation, discussing the curves provides a better means of comparing schemes than arbitrarily chosen times or accuracies (particularly when selected to make a new solution appear strong). In most cases, we expect the focus to be on accuracies close to the accuracy attained by an uncompressed baseline (which is typically desired). But in practice, as we later verify empirically, not all compression schemes can meet all accuracy targets, as compression often results in lower final model accuracy than the uncompressed baseline. Typically, for research comparing different schemes, we suggest that different techniques be run until convergence (*e.g.*, according to an early stopping standard [39]) to produce the TTA curves.

End-to-end evaluation is inevitably costly but necessary for gradient compression research, as focusing solely on metrics that are not end-to-end (such as training throughput) results in an insufficient understanding of the system performance and incomplete conclusions. However, for tasks such as parameter tuning, there are useful cheaper proxies that (combined with training throughput) quickly deliver a rough idea of the convergence speed at the beginning of the training. One example is vector's normalized mean squared error (vNMSE) [17, 29, 34, 54, 55], which measures the compression error between the true gradients' average and its estimate from the compressed gradients.

We note that some other end-to-end metrics could be a better fit in certain circumstances, such as cost-to-accuracy and power-to-accuracy. There may be cases where one is willing to use a less robust metric, such as by picking a specific target accuracy (e.g., 95%), but this is generally not a suitable goal when comparing compression schemes generally (as in research), and should be justified when done.

Task	TF32+FP16	TF32+FP32	FP32+FP16	FP32+FP32
BERT	3.32	2.44	3.17	2.36
VGG19	9.31	6.59	8.73	6.37

Table 2: Throughput (in rounds per second) of baselines varying training and communication precision. FP32+FP16 means we train with FP32 and communicate in FP16 precision. TF32, FP32, FP16 refer to Tensorfloat [6], full- and half-precision.

The choice of baselines. Most of the literature ([11, 14, 23, 30, 34, 36, 57, 60, 62]) compares against a baseline using full precision (FP32) aggregation. However, the use of half-precision (FP16) [1], which has gained wide hardware support [8, 26], provides a stronger baseline. It achieves 50% less communication and thus much higher throughput (see Table 2), and as shown in our experiments, the accuracy degradation is negligible.

3 A CASE STUDY

In this section, we conduct a case study to better elaborate on the design and evaluation issues of gradient compression. We focus on three common categories of gradient compression method: sparsification [12, 30, 32, 51] (§3.1), quantization [13, 18, 34, 63] (§3.2) and low-rank decomposition [57, 58] (§3.3). Namely, as representatives, we use TopK sparsification [12, 51], THC quantization [34] and PowerSGD decomposition [57]. We also introduce strategic design changes to these techniques, showing how considering issues motivates better design of gradient compression.

Our prototypes are implemented in NCCL [10] and PyTorch DDP [35]. We choose two common tasks for evaluation, namely VGG19 [50] for classification on TinyImageNet [31] and BERT-large [22] for masked language modeling [40] on WikiText-103 [37]. The per-worker batch size is set as 32 and 4, respectively. The prototypes are deployed on a testbed with two nodes, each equipped with two NVIDIA A100 GPUs [5] (for a total of 4 GPUs) and a Mellanox ConnectX-6 100Gbps NIC [3]. Early stopping [39] is applied to terminate training upon model convergence. We record the testing perplexity for BERT-large and the top-1 accuracy for VGG19. To plot the TTA figures, we apply a rolling average over 3750 rounds (0.3 epochs) for BERT-large and 7810 rounds (10 epochs) for VGG19.

Notation	Meaning
b	Volume of all-reduce communication in bits per coordinate.
n	The number of workers.
d	The total number of coordinates in each gradient vector.
K	The number of top K coordinates selected to be aggregated.
C	The chunk size of TopKC which gradients are partitioned into.
J	The number of top chunks selected for aggregation in TopKC.
J'	The total number of coordinates in top J chunks, *i.e.*, $J' = JC$.
q	The q-bit integers into which each FP32 gradient is quantized.
l	Number of RHT iterations. Gradients are padded to size 2^l.
l'	The number of iterators for partial rotation. $l' \leq l$.
$Sat(\cdot,\cdot)$	The saturation operator.
r	The target matrix rank r for PowerSGD.

Table 3: Notation used in our case study.

We measure the communication overhead in terms of the all-reduce input size, in bits per coordinate denoted by b. Note that the actual bandwidth required depends on the specific all-reduce scheme. In ring-all-reduce, the total overhead is roughly $2b$ per coordinate (reduce-scatter and all-gather).

3.1 TopK Sparsification

3.1.1 Background and issues.

Sparsification compresses by only sending some of the gradient coordinates. Intuitively, one would want to send the top K coordinates with the largest *aggregated* (summed across gradients) value, which we term Global TopK. Unfortunately, finding these indices is challenging without computing the aggregated gradient to begin with. Thus, in practical (local) TopK sparsification (e.g., [12, 47, 51]) implementations, each worker extracts its K largest (in absolute value) coordinates and their indices to be aggregated later. Prior works typically use TopK compression with the less scalable PS architecture or a distributed all-gather collective [9, 41] as their aggregation scheme but do not implement it with all-reduce.

Computational overhead. The TopK selection operation and rearrangement of coordinates is a major bottleneck [45] (see §3.1.3). The reason is that they involve non-consecutive memory accesses with poor locality, which slows down the processing speed of the GPU [38].

All-reduce incompatibility. With a standard all-reduce operation without compression with n workers and gradients of size d, each worker sends and receives d/n-sized blocks at each of $2(n-1)$ steps, and these blocks are summed (reduced) at intermediate workers. When implementing TopK in a distributed setting, a challenge is that the coordinates sent by each worker may be different, resulting in up to nK distinct coordinates, which may not give significant compression if nK is comparable to or even larger than d.

3.1.2 Our proposed improvement: Chunks.

Overview. We propose a TopK variant dubbed TopK Chunked (TopKC in short) that has less computational overhead and is compatible with all-reduce to approximate Global TopK. The key idea is to perform a lightweight step to reach consensus on a good set of coordinates to aggregate (that are not the top K coordinates). In TopKC, for a hyperparameter C, each worker partitions its gradient into $\lceil d/C \rceil$ fixed-sized chunks. The workers then decide which chunks to communicate with an initial all-reduce communication round. In this round, each worker sends the squared norm of each chunk, allowing all workers to agree on the $J < d/C$ chunks with the largest sum of L2 norms. Intuitively, chunks with large sum-of-norms are likely to contain large coordinates that are worth aggregating. We then aggregate these agreed-upon chunks using all-reduce, thus making TopKC all-reduce-compatible. A benefit of TopKC is that our memory access pattern is mainly sequential and the expensive top-K calculation [45] operates on fewer values than TopK ($\lceil d/C \rceil$ rather than d), allowing faster execution in practice.

Compression	$b = 0.5$	$b = 2$	$b = 8$
TopKC	0.273	0.142	0.0280
TopKC Permutation	0.398	0.297	0.123

Table 4: vNMSE of the aggregated gradients with TopKC and TopKC with random permutation for BERT.

We show that TopKC can be an effective heuristic, in part because of *spatial locality*, meaning large coordinates tend to appear closer to each other. To demonstrate spatial locality, we compare against a variant, TopKC Permutation, that randomly permutes the coordinates, eliminating any spatial locality. We choose to use vNMSE [55] (Section 2.2) to study the compression error, and follow the experimental settings shown in Section 3.1.3. Table 4 indicates that TopKC takes advantage of spatial locality, since it performs significantly better than when the coordinates are randomly permuted.

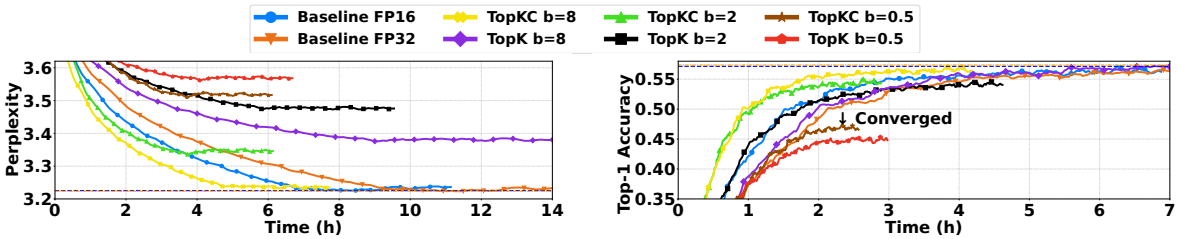
Design changes. TopKC consists of the following steps.

1) We partition each local gradient into smaller chunks of size C. Let $c_p^{(i)}$ be the pth chunk for the ith worker, so each has a vector of values $c_1^{(i)} \cdots c_{\lceil d/C \rceil}^{(i)}$. Each worker then calculates the squared L2 norm of each chunk, $||c_p^{(i)}||_2^2$. The squared L2 norms (in half-precision) are then aggregated across workers for each chunk using all-reduce, *i.e.*, $s_p := \sum_i ||c_p^{(i)}||_2^2$.

2) The consensus TopK chunks with the highest s_p values are determined locally by all workers. These will be the global TopK chunks. The sparsified gradients containing the global TopK chunks are summed via an all-reduce communication in *half-precision* (i.e., FP16).

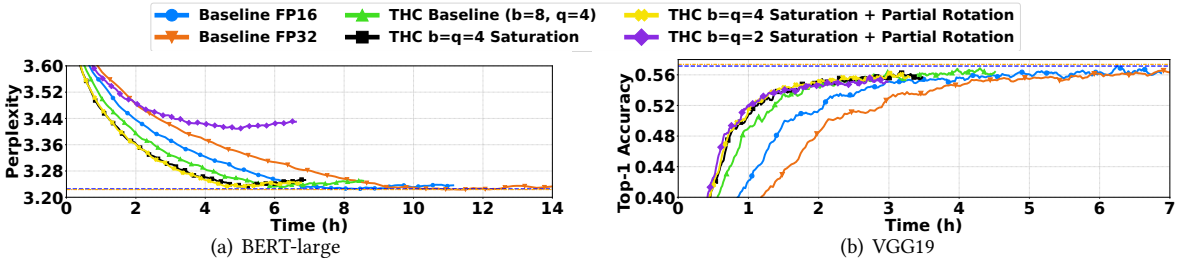
The first stage of determining the local squared L2 norms can be computed efficiently, and all-reducing the squared L2 requires communication of $16/C$ bits per coordinate. Adding the second stage of all-reduce communication, the total cost in bits per coordinate is $b = 16(JC/d + 1/C)$, where J denotes the number of TopK chunks selected each round.

We believe our chunk-based aggregation approach, which allows us to cheaply coordinate the coordinates for sparsification, may be generalizable to other schemes [30, 49].



(a) BERT-large (b) VGG19

Figure 1: The TTA (rolling averaged) of our TopK Chunked (TopKC) solution compared with TopK and the baselines. The dashed lines indicate the converged perplexity/accuracy for Baseline FP16 and Baseline FP32 respectively. The training of each method stops after a given number of epochs (and not hours) after convergence.



(a) BERT-large (b) VGG19

Figure 2: The TTA of THC's simple adaptation to all-reduce compared with THC adding saturation and partial rotation.

Task	Compression	$b = 0.5$	$b = 2$	$b = 8$
BERT-large	TopK	5.53	3.87	2.50
(345M params)	TopKC	6.06	6.02	4.78
VGG19	TopK	21.5	13.9	7.60
(144M params)	TopKC	24.9	22.2	15.2

Table 5: Throughput (in rounds per second) comparing the original TopK against TopK Chunked (TopKC). b refers to the bits-per-coordinate as the aggregation input of the all-gather collective for TopK and all-reduce for TopKC. Note that the final accuracy is reflected in TTA curves of Figure 1.

Task	$b = 0.5$	$b = 2$	$b = 8$
BERT	9.7%	12.5%	8.7%
VGG19	11.9%	12.1%	8.2%

Table 6: The compression overhead (the percentage of time spent on the computationally heavy components) of TopK.

3.1.3 Preliminary evaluation.

Setup. For both TopK and TopKC, we keep the bits-per-coordinate b the same, varying from $b = 0.5$ to 8. For TopK, we follow its typical implementations [28, 48] and transmit FP16-compressed values and their 32-bit indices of the top K coordinates. [2] Thus, we need to send $b = (48K)/d$ bits per coordinate. An all-gather collective is chosen for aggregation. For TopKC, we set $C = 64$ for $b = 8$ and $b = 2$, and $C = 128$ for $b = 0.5$. Denoting by $J' = JC$ the total number of coordinates that belong to one of the top J chunks, $J' > K$ holds given the same b, thanks to the fact that TopKC saves much communication on exchanging indices for the above parameters. Finally, error-feedback [29, 44] is applied to both TopK and TopKC.

[2]We note that it is possible to use 16-bit indices, e.g., by using delta-encoding and including additional coordinates to ensure the differences between consecutive indices are representable with 16 bits each. However, its GPU-unfriendly computation means that the TTA may not improve, and this does not seem to be how TopK is implemented in practice.

Compression	$b = 0.5$	$b = 2$	$b = 8$
TopK	0.303	0.185	0.0865
TopKC	0.273	0.142	0.0280

Table 7: vNMSE of aggregated gradients comparing TopKC and TopK for BERT with respect to bits per coordinate b.

Evaluation issues. First, as illustrated in Figure 1 and Table 2, FP16 appears as a stronger baseline in terms of both TTA and throughput. It is important to compare with the stronger FP16 baseline, as TopK $b = 8$ marginally outperforms FP32 in VGG19 but falls behind FP16's TTA.

Second, we see that training throughput can be a misleading metric for end-to-end performance. For example, for both TopKC and TopK, reducing b from 8 to 0.5 enhances training throughput in BERT-large but leads to degraded TTA and final accuracy. This difference can be attributed to the increased compression error, as displayed in Table 7.

Analysis of our design changes. As shown in Figure 1, our TopKC variation has better TTA results than TopK, and this is due to both higher throughput and better accuracy. TopKC's throughput outperforms the original TopK by up to 2× (Table 5). The improvement can be primarily attributed to TopKC's compatibility with all-reduce, which is more communication-efficient than all-gather. While TopKC's computational overhead is negligible, even with minimal communication overhead (e.g., $b = 0.5$), Table 6 shows that TopK's

Task	#bits	Full Rotation	Partial Rotation	No Rotation
BERT	Sat, $b = q = 2$	5.59	5.75	5.84
	Sat, $b = q = 4$	5.37	5.47	5.54
	BL $b = 8, q = 4$	4.32	N/A	N/A
VGG19	Sat, $b = q = 2$	19.9	21.5	22.7
	Sat, $b = q = 4$	18.4	19.4	20.3
	BL $b = 8, q = 4$	14.2	N/A	N/A

Table 8: Throughput of THC with Saturation (Sat) compared with the baseline (BL) which adds 4 more communication bits to prevent overflows during aggregation (so $b = 8$). Note that the final accuracy is reflected in TTA curves of Figure 2.

computation takes ∼ 10% of the training time. In terms of the compression error, as indicated in Table 7, TopKC outperforms TopK in terms of vNMSE. This can be attributed to TopKC aggregating more coordinates than TopK given the same b, *i.e.*, $J' > K$.

3.2 THC Quantization

3.2.1 *Background and issues.*

THC [34] is a quantization-based compression algorithm designed specifically for PS architectures, where the PS can optionally be offloaded to programmable switches [4, 19]. It adopts stochastic quantization to map floating-point gradients into q-bit integers $[0, 2^q - 1]$. The value range between the minimum and maximum gradient values is equally split into subranges with quantized values at the boundaries, and each coordinate is stochastically rounded to one of the two nearest quantized values. One key optimization for THC to enhance quantization accuracy is to adopt the Randomized Hadamard Transform (RHT) [25], which rotates the gradient randomly before quantization and thus decreases the range between the minimum and maximum gradient values.

Computational overhead. We find that RHT incurs considerable overhead. As shown in Table 8, THC with RHT is 4.4% and 13.2% lower in training throughput than THC without a rotation for BERT and VGG19 tasks respectively, suggesting improvement is possible if RHT can be made more efficient without a significant loss of accuracy. The high computational overhead arises because RHT requires $O(d \log d)$ steps to compute and can have poor locality of memory accesses. The latter is because RHT involves memory-distant operations; for large d it could not fit into the fast but small shared memory of GPUs, and fallbacks to the slower global memory.

All-reduce incompatibility. The aggregation process of a coordinate p can be formulated as summing up quantized values in $[0, 2^q - 1]$ from n workers, *i.e.*, $\sum_i g_p^{(i)}$ where $g_p^{(i)}$ denotes the quantized value of coordinate p for worker i. If we allocate $b = q$ bits per coordinate for aggregation, the summed integer values could overflow the value range that q bits can represent. This is not an issue for the PS architecture, where the PS is the destination for a full aggregation and can simply allocate more bits on the server to prevent overflows. For all-reduce, however, an intermediate hop has to transmit partially aggregated gradients, which could overflow. To achieve all-reduce compatibility, THC suggests a simple adaptation by increasing the number of bits $b \geq q$ for communication to accommodate the increasing value range. This leads to extra communication and is still not scalable for a larger number of workers.

3.2.2 *Our proposed improvements.*

Partial rotation. To alleviate the computational overhead of RHT, we observe that its recursive structure allows us to introduce an early stopping condition that takes into account the shared memory size of GPUs. Namely, for a vector of size 2^l, the full RHT involves l iterations. Instead, we stop the transform after $l' \leq l$ iterations, picking the largest l' such that the shared memory size is larger than $2^{l'}$. This is mathematically equivalent to dividing the gradient into $2^{l'}$-sized chunks and rotating each separately but with the advantage of using only a single GPU kernel to do so.

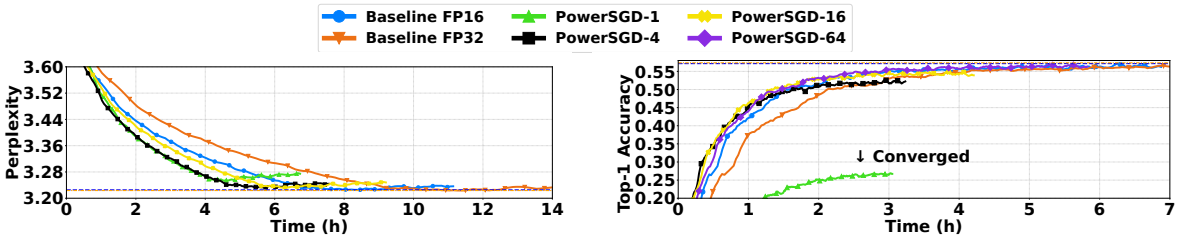
After the partial rotation, we compute the value range $\max_p - \min_p$ of each chunk p, so that coordinates of higher magnitude only affect the quantization precision locally. Evaluation (Section 3.2.3) shows that partial rotation achieves faster compression while remaining effective in reducing the value range to improve quantization precision.

Saturation-based aggregation. To achieve better all-reduce compatibility and handle overflows, we propose a saturation-based lossy aggregation that enables aggregation without extra communication overhead. One insight is that RHT transforms the gradients $\nabla^{(i)}$ to a vector whose entries roughly follow a normal distribution $\mathcal{N}(0, 1/||\nabla^{(i)}||_2^2)$ [55]. That is, the coordinates are strongly concentrated around 0 and may cancel each other to some extent during the summation, reducing the chance of overflow.

We thus propose replacing summing gradients with the saturation operator $\mathrm{Sat}(\cdot,\cdot)$ acting on top of the rotated and normalized gradients. Formally, $\mathrm{Sat}(x, y) = \min(2^{b-1} - 1, \max(-2^{b-1} + 1, x + y))$, where x, y are b-bit for aggregation and $b = q$ in our experiments. We find that saturation does not introduce much error in our experiments because of the low probability of overflows after rotation and normalization. Other setups, e.g., using a large number of workers, may affect this conclusion.

As the number of workers n increases, our saturation-based aggregation has to allocate more communication bits $b > q$ to upper-bound the probability of overflows. However, the number of bits for saturation is generally less than the simple solution, as it takes advantage of the cancellation that arises from positive and negative values.

Our techniques may generalize to other quantization schemes, e.g., addressing integer summation overflow through saturation for [13, 18, 63] and enhancing speed by replacing full RHT with partial rotation, e.g., for [52, 55].



(a) BERT-large (b) VGG19

Figure 3: The TTA of PowerSGD, altering the matrix rank r.

Task	$r=1$		$r=4$		$r=16$		$r=64$	
	b	Thr.	b	Thr.	b	Thr.	b	Thr.
BERT	0.0797	5.49	0.217	4.89	0.764	4.01	2.95	3.03
VGG19	0.0242	21.0	0.0872	19.8	0.339	15.2	1.36	11.0

Table 9: Bits-per-coordinate and throughput (in rounds per second) for PowerSGD, varying the rank r.

3.2.3 Preliminary evaluation. We now present evaluations of our proposed optimization of THC, setting $b = q = 4$. We compare against baseline THC that deploys full rotation of RHT and avoids overflows by using $b = 8$ bits per coordinate.

Analysis of our design changes. Table 8 shows that our proposed optimization effectively improves the throughput. First, the adoption of partial rotation contributes an up to 5.5% increase in throughput ($b = 4$), showing the successful reduction of the computational overhead. Second, the saturation-based aggregation achieves 50% less communication, bringing up to 29.6% higher throughput.

Accordingly, the benefits of higher throughput are reflected in the improved end-to-end TTA results, as depicted in Figure 2. In particular, by adding saturation and partial rotation to the original THC, TTA converges progressively faster to reach a given perplexity or top-1 accuracy. This, combined with the indistinguishable final perplexity/accuracy difference with the baselines, supports that both saturation and partial rotation incur little accuracy degradation, so the increase in throughput also leads to a better TTA.

Finally, we reduce the communication overhead by setting $b = q = 2$. According to Table 8 and Figure 2, the throughput is improved from $b = 4$, but the TTA on BERT significantly degrades even compared to the Baseline FP16. This provides another piece of evidence that solely measuring the throughput is not adequate in evaluating end-to-end performance.

3.3 PowerSGD Low-rank Decomposition

We now analyze PowerSGD, a compression scheme based on low-rank decomposition. It is parameterized with a small integer r indicating the target rank of approximated matrices. The results are shown in Figure 3 and Table 9. Since PowerSGD is compatible with all-reduce [11], we instead discuss the issues of computational overhead and evaluation.

Computational overhead. In Table 9, we find that PowerSGD could achieve high com- pression ratios, with up to 47× (for $r = 16$) less bits-per-coordinate than FP16. However, such a considerable communication reduction does not improve as much throughput as other compression schemes (Table 5 and 8). Moreover, increasing r from 1 to 64 slows down the throughput by nearly half, with the communication overhead still being negligible. Profiling reveals that the major bottleneck is the *overwhelmingly expensive* operation of matrix orthogonalization [24], which consumes 39.7% and 47.4% of the training time for BERT and VGG19 with $r = 64$.

The choice of metrics and baselines. The results as depicted in Figure 3 provide another example of how $r = 1$ achieves a higher throughput than $r = 16$ and 64 but converges slower and to a lower accuracy for VGG19, highlighting the importance of TTA evaluation. Also, PowerSGD $r = 4$ significantly outperforms Baseline FP32 but only provides a modest benefit compared to Baseline FP16, highlighting the importance of using a stronger baseline.

4 CONCLUSIONS AND DISCUSSION

This paper provides insights into the issues involved in designing and evaluating state-of-the-art gradient compression systems. By considering the challenges in measuring performance, going beyond simple but insufficient measures such as throughput to the more robust measure of time to accuracy, we aim to shed light on designs that enhance the utility of gradient compression. Specifically, our case study describes how we identify problems and solve them with this framework in mind.

We note, however, that perhaps time to accuracy is itself not the only appropriate metric. Key considerations not taken into account with TTA are the overall power used or the overall cost to construct the model. We leave the issue of determining a framework that takes power and/or cost into account as an exciting future direction. We believe the approach suggested here, of focusing on the right metrics to judge end-to-end performance, would be very important to comparing compression schemes in these contexts.

Acknowledgments: We thank the anonymous reviewers for their insightful comments and suggestions. Michael Mitzenmacher was supported in part by NSF grants CCF-2101140, CNS-2107078, and DMS-2023528.

REFERENCES

[1] 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008), 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

[2] 2017. Ring all reduce. https://github.com/baidu-research/baidu-allreduce

[3] 2018. The NVIDIA® Mellanox® ConnectX®-6 SmartNIC. https://www.nvidia.com/en-sg/networking/ethernet/connectx-6/

[4] 2020. Intel® tofino™ series programmable ethernet switch asic. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[5] 2020. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-gb/data-center/a100/

[6] 2020. TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20. https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/

[7] 2023. Memory Footprint and FLOPs for SOTA Models in CV/NLP/Speech. https://github.com/amirgholami/ai_and_memory_wall/blob/main/imgs/pdfs/ai_and_compute.pdf

[8] 2023. Train with mixed precision. https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

[9] 2024. Collective operations in NCCL. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html

[10] 2024. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl

[11] Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, and Dimitris Papailiopoulos. 2022. On the utility of gradient compression in distributed training systems. *Proceedings of Machine Learning and Systems* 4 (2022), 652–672.

[12] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse Communication for Distributed Gradient Descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 440–445.

[13] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in neural information processing systems* 30 (2017).

[14] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, Ruichuan Chen, and Yinlong Xu. 2021. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 359–375.

[15] Ran Ben-Basat, Yaniv Ben-Itzhak, Michael Mitzenmacher, and Shay Vargaftik. 2024. Optimal and Approximate Adaptive Stochastic Quantization. In *NeurIPS*.

[16] Ran Ben Basat, Michael Mitzenmacher, and Shay Vargaftik. 2021. How to Send a Real Number Using a Single Bit (And Some Shared Randomness). In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*.

[17] Ran Ben-Basat, Amit Portnoy, Gil Einziger, Yaniv Ben-Itzhak, and Michael Mitzenmacher. 2024. Accelerating Federated Learning with Quick Distributed Mean Estimation. In *ICML*.

[18] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. 2018. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*. PMLR, 560–569.

[19] Mihai Budiu and Chris Dodd. 2017. The P416 Programming Language. *SIGOPS Oper. Syst. Rev.* 51, 1 (Sept. 2017), 5–14. https://doi.org/10.1145/3139645.3139648

[20] Xiaoqi Chen, Shay Vargaftik, and Ran Ben-Basat. 2024. When ML Training Cuts Through Congestion: Just-in-Time Gradient Compression via Packet Trimming. In *Hotnets*.

[21] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[23] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. 2021. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 676–691.

[24] Walter Gander. 1980. Algorithms for the QR decomposition. *Res. Rep* 80, 02 (1980), 1251–1268.

[25] A Hedayat and Walter Dennis Wallis. 1978. Hadamard matrices and their applications. *The annals of statistics* (1978), 1184–1238.

[26] Nhut-Minh Ho and Weng-Fai Wong. 2017. Exploiting half precision arithmetic in Nvidia GPUs. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[27] X Jia. 2018. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205* (2018).

[28] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. 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)*. 463–479.

[29] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. 2019. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*. PMLR, 3252–3261.

[30] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. 2019. Parallax: Sparsity-aware data parallel training of deep neural networks. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–15.

[31] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[32] Haoyu Li, Yuchen Xu, Jiayi Chen, Rohit Dwivedula, Wenfei Wu, Keqiang He, Aditya Akella, and Daehyeok Kim. 2024. Accelerating Distributed Deep Learning using Lossless Homomorphic Compression. *arXiv preprint arXiv:2402.07529* (2024).

[33] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*. 583–598.

[34] Minghao Li, Ran Ben Basat, Shay Vargaftik, ChonLam Lao, Kevin Xu, Michael Mitzenmacher, and Minlan Yu. 2024. {THC}: Accelerating Distributed Deep Learning Using Tensor Homomorphic Compression. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1191–1211.

[35] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[36] Ahmed M Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, and Marco Canini. 2021. An efficient statistical-based gradient compression technique for distributed training systems. *Proceedings of Machine Learning and Systems* 3 (2021), 297–322.

[37] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*

(2016).

[38] NVIDIA. 2024. CUDA memory hierarchy. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html?highlight=memory%20hierarchy#memory-hierarchy

[39] Lutz Prechelt. 2002. Early stopping-but when? In *Neural Networks: Tricks of the trade*. Springer, 55–69.

[40] Julian Salazar, Davis Liang, Toan Q Nguyen, and Katrin Kirchhoff. 2019. Masked language model scoring. *arXiv preprint arXiv:1910.14659* (2019).

[41] Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. 2019. *Sequential and Parallel Algorithms and Data Structures*. Springer.

[42] Peter Sanders, Jochen Speck, and Jesper Larsson Träff. 2009. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.* 35, 12 (2009), 581–594.

[43] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtárik. 2021. Scaling distributed machine learning with {In-Network} aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 785–808.

[44] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 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*.

[45] Anil Shanbhag, Holger Pirk, and Samuel Madden. 2018. Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*. 1557–1570.

[46] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. {FlexTOE}: Flexible {TCP} Offload with {Fine-Grained} Parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 87–102.

[47] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. 2019. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772* (2019).

[48] Shaohuai Shi, Qiang Wang, Kaiyong Zhao, Zhenheng Tang, Yuxin Wang, Xiang Huang, and Xiaowen Chu. 2019. A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2238–2247.

[49] Shaohuai Shi, Xianhao Zhou, Shutao Song, Xingyao Wang, Zilin Zhu, Xue Huang, Xinan Jiang, Feihu Zhou, Zhenyu Guo, Liqiang Xie, et al. 2021. Towards scalable distributed training of deep learning on public cloud clusters. *Proceedings of Machine Learning and Systems* 3 (2021), 401–412.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[51] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with memory. *Advances in neural information processing systems* 31 (2018).

[52] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed mean estimation with limited communication. In *International conference on machine learning*. PMLR, 3329–3337.

[53] Zhenheng Tang, Shaohuai Shi, Wei Wang, Bo Li, and Xiaowen Chu. 2020. Communication-efficient distributed deep learning: A comprehensive survey. *arXiv preprint arXiv:2003.06307* (2020).

[54] Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben Itzhak, and Michael Mitzenmacher. 2022. Eden: Communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*. PMLR, 21984–22014.

[55] Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. 2021. Drive: One-bit distributed mean estimation. *Advances in Neural Information Processing Systems* 34 (2021), 362–377.

[56] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G Andersen, Gregory R Ganger, Garth A Gibson, and Brian Mueller. 2009. Safe and effective fine-grained TCP retransmissions for datacenter communication. *ACM SIGCOMM computer communication review* 39, 4 (2009), 303–314.

[57] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. 2019. PowerSGD: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems* 32 (2019).

[58] Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. 2018. Atomo: Communication-efficient learning via atomic sparsification. *Advances in neural information processing systems* 31 (2018).

[59] Hao Wang, Han Tian, Jingrong Chen, Xinchen Wan, Jiacheng Xia, Gaoxiong Zeng, Wei Bai, Junchen Jiang, Yong Wang, and Kai Chen. 2024. Towards {Domain-Specific} Network Transport for Distributed {DNN} Training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1421–1443.

[60] Zhuang Wang, Haibin Lin, Yibo Zhu, and TS Eugene Ng. 2023. Hi-Speed DNN Training with Espresso: Unleashing the Full Potential of Gradient Compression with Near-Optimal Usage Strategies. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 867–882.

[61] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. 2023. {SRNIC}: A scalable architecture for {RDMA}{NICs}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1–14.

[62] Zhuang Wang, Xinyu Crystal Wu, Zhaozhuo Xu, and TS Eugene Ng. 2023. CUPCAKE: A Compression Optimizer for Scalable Communication-efficient Distributed Training. In *Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys' 23)*. Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys' 23).

[63] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in neural information processing systems* 30 (2017).

[64] Hang Xu, Chen-Yu Ho, Ahmed M Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. 2021. Grace: A compressed communication framework for distributed machine learning. In *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*. IEEE, 561–572.