L-GreCo: Layerwise-Adaptive Gradient Compression for Efficient Data-Parallel Deep Learning

Ilia Markov [1 *] **Kaveh Alimohammadi** [1 *] **Elias Frantar** [1] **Dan Alistarh** [1 2]

Abstract

Data-parallel distributed training of deep neural networks (DNN) is pervasive, but can still experience communication bottlenecks. To address this, several compression mechanisms such as quantization, sparsification, and low-rank approximation have been introduced. Despite this progress, most implementations are sub-optimal, as they apply compression uniformly across DNN layers, although layers are heterogeneous in terms of parameter count and their impact on accuracy. In this work, we provide a general framework for dynamically adapting the degree of compression across the model's layers during training, improving overall compression while leading to substantial speedups, without sacrificing accuracy. Our framework, called `L-GreCo`, is based on an adaptive algorithm that automatically picks the optimal compression parameters for model layers, guaranteeing the best compression ratio, while satisfying an error constraint. Extensive experiments over image classification and language modeling tasks show that `L-GreCo` is effective across *all existing families of compression methods*, and achieves up to 2.5× training speedup and up to 5× compression improvement over efficient implementations of existing approaches, and can even complement existing adaptive algorithms. An anonymized implementation is available at `https://github.com/LGrCo/L-GreCo`.

1 Introduction

The growth in model and dataset sizes for deep learning has made distribution a standard approach to training. The most popular strategy is *synchronous data-parallel distribution*, which splits the data between parallel workers, each of which computes stochastic gradients over their data, and then averages the workers' gradients in a synchronous step. This procedure has several advantages, but induces two main overheads: The *synchronization cost* of barrier-like synchronization at every step, and the *communication cost* of exchanging the gradients in an all-to-all fashion.

A popular approach for reducing the cost of gradient communication, which is the main focus of our paper, is *lossy gradient compression* (Seide et al., 2014; Alistarh et al., 2017; Dryden et al., 2016; Vogels et al., 2019), which reduces the number of communicated bits per iteration. Hundreds of such techniques have been proposed, which can be roughly categorized into three method families. The first is *quantization* (Seide et al., 2014; Alistarh et al., 2017; Wen et al., 2017), which reduces the bit width of the communicated gradients in a variance-aware fashion in order to preserve convergence. The second is *sparsification* (Strom, 2015; Dryden et al., 2016; Lin et al., 2017), reducing the number of gradient components updated at every step, which are chosen via various saliency metrics. The third and most recent approach is *low-rank approximation* (Wang et al., 2018; Vogels et al., 2019), which leverages the low-rank structure of gradient tensors to minimize communication.

In practice, these approaches come with trade-offs in terms of compression versus ease of use. For instance, gradient quantization is easy to implement and deploy, but only provides limited compression before accuracy degradation; sparsification and low-rank approximation can provide order-of-magnitude compression improvements, but come with additional costs in terms of maintaining error correction and careful hyper-parameter tuning. These trade-offs have been investigated via *adaptive* compression methods (Agarwal et al., 2021; Markov et al., 2022; Faghri et al., 2020), which adjust the compression to the error incurred during various phases of DNN training.

Currently, there is still a significant gap between ideal, theoretically-justified compression methods, and their efficient systems implementations. For example, the theory of gradient compression (Karimireddy et al., 2019; Nadiradze et al., 2021) suggests that the ideal compression method in terms of convergence is that which minimizes the *norm of the compression error*, i.e., the difference between the global model gradient and the compressed one. (For instance, global TopK selection provides the best sparsification-based compressor for a given parameter K (Sahu et al., 2021).) Yet, parameter selection based on a global gradient at each step is not practical from the systems perspective: performing global selection, such as TopK, requires waiting for the whole model gradient to be available; yet fast implementa-

tions of data-parallel training transmit each layer's gradients as soon as they are generated, overlapping communication and computation. Moreover, many existing implementations miss significant opportunities for optimization: modern models such as Transformers (Vaswani et al., 2017) are highly heterogeneous in terms of both layer sizes and layer sensitivity to gradient compression (see Figure 1), and gradient compression impacts various stages of training differently (Achille et al., 2018).

Thus, there is still a gap between the theory and practice of gradient compression, leading some to question the usefulness of this approach (Agarwal et al., 2022). In this paper, we address this gap and show that, when paired with efficient and adaptive systems support, gradient compression can be a powerful technique for efficient data-parallel training. Specifically, the question we address is the following: Given an arbitrary model and gradient compression technique, is there an efficient way to balance accuracy constraints, such as *layer sensitivities*, and the communication constraints, such as *layer sizes*, dynamically during training in order to maximize speedup, without sacrificing theoretical convergence and practical accuracy?

To address this, we introduce `L-GreCo`, an efficient and general framework for Layer-wise parametrization of GRadiEnt COmpression. At the algorithmic level, L-GreCo is based on a new formalization of the *layer-wise adaptive compression* problem, which identifies per-layer compression parameters, e.g. per-layer sparsity or quantization levels, seeking to maximize compression, under a fixed constraint on the total compression error, which ensures both good theoretical convergence, and negligible accuracy loss. At the system level, `L-GreCo` works by integrating with standard training frameworks, such as `torch.distributed`, to exploit model heterogeneity in terms of both per-layer structure and per-layer sensitivity, determining on-the-fly by how much to compress individual layer gradients in order to maximize compression or end-to-end training times.

We validate `L-GreCo` across *all existing families of compression strategies*: quantization, sparsification, and low-rank compression, across a variety of vision and language tasks. `L-GreCo` consistently achieves higher compression rates than existing manual or adaptive strategies (Vogels et al., 2019; Agarwal et al., 2021), in a black-box fashion, and is particularly effective for modern Transformer models, in both single- and multi-node settings.

We summarize our contributions as follows:

- We introduce a new approach leveraging the heterogeneous structure of DNNs in order to reduce communication overheads while maintaining convergence, guaranteeing optimal layer-wise compression-based by balancing a theoretically-justified error metric with an optimization objective maximizing compression.
- We provide an extensive empirical evaluation on different neural networks (ResNet18, ResNet50, Transformer-XL, Transformer-LM) with different datasets (CIFAR-100, ImageNet, Wikitext-103) showing that `L-GreCo` reduces communication by up to 5× and achieves speedups up to 2.5× without loss of accuracy or significant tuning, across both single and multi-node deployments.
- In addition, we conduct the first in-depth study of both sensitivity and performance metrics. We show that the theoretically-justified error norm metric is essentially equivalent to more complex metrics based on examining output loss. From the performance perspective, we show that optimization objectives seeking to maximize absolute compression lead to similar results to objectives that minimize transmission time.
- Finally, we show that `L-GreCo` is compatible with prior adaptive compression schemes; specifically, it can be extended to use information about the different stages of training (Agarwal et al., 2021), leading to further performance improvements.

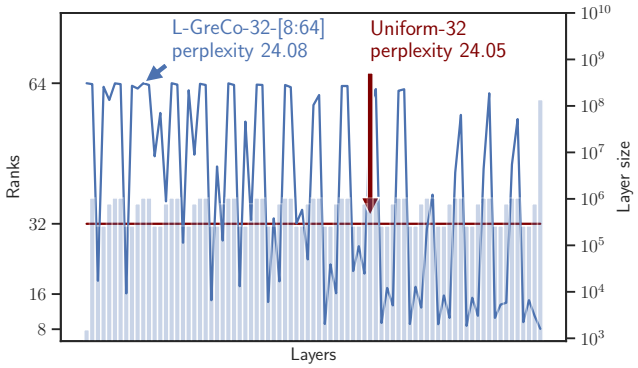


Figure 1. Profile of `L-GreCo` rank choices for PowerSGD compression on Transformer-XL. The red line represents uniform compression, while the blue line represents the `L-GreCo` profile. Transparent bars show layer sizes. Layers are indexed in the order they are communicated. The annotated number is the final test perplexity (ppl) for the experiment (lower is better). Here, the average compression of `L-GreCo` is **1.5x higher** than uniform.

2 Related Work

Compression methods. Gradient compression usually employs three strategies: quantization, sparsification, and low-rank decomposition. Quantization methods (Seide et al., 2014; Alistarh et al., 2017; Wen et al., 2017; Lim et al., 2018; Ramezani-Kebrya et al., 2021) use lower precision of each gradient component, reducing the number of transmitted bits. They are easy to implement and work under stable

hyper-parameters (Alistarh et al., 2017; Xu et al., 2021; Markov et al., 2022). However, their compression is limited by the fact that at least one bit per entry must usually be transmitted. Sparsification techniques (Strom, 2015; Dryden et al., 2016; Lin et al., 2017; Karimireddy et al., 2019) circumvent this by identifying *salient components* in the gradient and only transmit such subsets. Finally, gradient decomposition algorithms (Vogels et al., 2019; Wang et al., 2018) use the fact that the layer-wise gradient tensors are known to be well-approximable via low-rank matrices and aim to design light-weight projection approaches that also provide low error. Sparsification and low-rank techniques usually require *error correction* buffers to preserve good convergence, as well as non-trivial hyper-parameter tuning. As we show experimentally, `L-GreCo` is compatible with all of these approaches and can provide significant additional bandwidth savings for each such strategy without sacrificing model accuracy and without tuning.

Adaptive Compression. The general idea of *adapting* the degree of compression during training has been investigated by AdaComp (Chen et al., 2018), which proposes a self-tuning adaptive compression method; yet, their method does not adapt compression parameters per layer and cannot be combined with other compression approaches. Faghri et al. (2020) adapts the quantization grid to the gradient distribution; yet, their approach is specifically tuned to quantization, and oblivious to layer heterogeneity.

Sahu et al. (2021) optimize *the total error over steps* for sparsification-based compression and suggest threshold global sparsifiers, which are shown to reach higher compression rates than uniform per-layer compression on small vision tasks (e.g. ResNet18 on CIFAR10/100). However, their approach is restricted to sparsification and leaves unclear how to tune the threshold for large-scale, sensitive models such as Transformers or ImageNet-scale models. In particular, we were unable to obtain good results with this approach on models such as Transformer-XL or Transformer-LM. In Section 5.3, we present a comparison with their approach on ResNet18/CIFAR-100, showing that our method yields both higher accuracy and higher compression.

Accordion (Agarwal et al., 2021) adapts the compression parameters of sparsification and low-rank compression based on the critical regimes of training. The algorithm alternates between two compression levels ("low" and "high"), provided by the user and is prone to accuracy loss. Our approach improves upon Accordion in terms of speedup, but also that we can combine our method with Accordion and obtain even higher gains. CGX (Markov et al., 2022) investigated a kmeans-based heuristic, which we show experimentally to be sub-optimal.

To our knowledge, our dynamic programming strategy has not been employed in the context of adaptive gradient compression. Related approaches have been investigated in the context of weight compression for DNNs, see e.g. (Wu et al., 2020; Aflalo et al., 2020; Frantar & Alistarh, 2022; Shen et al., 2022). Yet, there are major differences: (1) the error metrics and speedup objectives are different in the case of weight compression; (2) we execute online at training time, which means that our algorithm has to be extremely efficient and adapt to dynamic inputs.

Recently, Agarwal et al. (2022) questioned the utility of gradient compression for distributed training. Yet, their study is limited in the sense that they only consider a limited subset of compression methods and focus on NCCL-type implementations on bandwidth-overprovisioned networks. By contrast, we consider a more flexible system implementation that allows to map layer parameters to different compression levels and show practically that L-GreCo can yield speedups both on commodity single-node GPU servers and on general multi-node systems.

3 Problem Formulation

Goals. Assuming we are given a DNN model $\mathcal{M}$ with L layers and a compression technique, we would intuitively like to find a choice of compression parameters c_ℓ, one for each layer $\ell \in \{1, 2, \ldots, L\}$ which would minimize a metric representing damage of the training quality introduced by compression while minimizing the total number of bits transmitted. Yet, this intuitive description leaves open a range of details, such as 1) the notion of layer-wise metric that corresponds to the compression effect for a given set of parameters; 2) the exact problem formulation, constraining the compression effect or the compression ratio; and 3) an efficient implementation of such an algorithm.

Sensitivity Metrics. Since choosing the right sensitivity metric is key for accuracy recovery, we have investigated two different approaches. First, the sensitivity of a layer to gradient compression can be measured by the impact on the model's loss. To evaluate this, we set up the following experiment: We saved model checkpoints at different stages of the uncompressed training. Then, we conducted multiple short runs (50 steps) with the same data starting from the checkpoint, varying compression parameters. We use the difference of loss with and without compression as the metric. Since we wish to observe the model's reaction to the compression of individual layers, we vary compression parameters for each layer, not compressing other layers' gradients. With that, we collect the differences of loss for each layer and the compression parameter as the metric.

Another approach is based on gradient compression theory, which shows formally that the *squared ℓ_2 compression error* is a good measure of the convergence impact of compression technique (Karimireddy et al., 2019; Nadiradze et al., 2021;

Sahu et al., 2021). Here, we aggregate gradients for the training, then compress the aggregated gradients for each compression parameter and for each layer individually, and use the magnitude of the error as a metric.

As shown in Section 5.4, the two approaches are strongly correlated, and the resulting optimal parameters are close to each other. However, the loss-based approach is less practical, as it requires *offline* evaluation, altering the original training pipeline and additional training time. By contrast, the error-based approach can be used *online* during training and has negligible overheads. Thus, throughout the paper, we use the L2 norm of the compression error as the main sensitivity metric.

The Constrained Optimization Problem. Given an error metric, we formalize the layer-wise compression optimization problem as follows. Given a model $\mathcal{M}$ with L layers $\ell \in \{1, 2, \ldots, L\}$ and a compression technique, providing a set of compression choices $\mathcal{C} = \{c^1, c^2, \ldots c^k\}$ for each layer. We emphasize that, for simplicity, we consider a single compression technique and the same compression choices/levels for each layer, but our approach would also work for different techniques being applied to the same model and heterogeneous compression choices.

In this context, our method receives as input an error function $error(\ell, c^j)$, which provides the L2 norm of the compression error at layer ℓ for compression choice c^j, and a function $size(\ell, c^\ell)$ which measures the transmission cost of layer ℓ for choice c^ℓ. In addition, we assume to be given a fixed maximal error threshold $\mathcal{E}_{\max}$ which the algorithm should not violate. Then, we wish to find a layer-wise setting of compression parameters $c_1, \ldots, c_L$ with the objective:

$$\text{minimize} \sum_{\ell=1}^{L} size(\ell, c^\ell) \text{ s.t. } \sum_{\ell=1}^{L} error(\ell, c^\ell) \leq \mathcal{E}_{\max}.$$

Practically, this formulation minimizes the total transmission cost for the gradient tensors under a maximum *additive* constraint on the gradient compression error. One implicit assumption is that the metric $error(\cdot, \cdot)$ is *additive* over layers and that it is possible to obtain a "reference" error upper bound, which does not result in accuracy loss. We will see that this is the case for the error metric we adopt.

The Error Bound. We pick the error bound $\mathcal{E}_{\max}$ to track that of a reference compression approach which is *known not to lose accuracy relative to the baseline*. Here, we leverage the fact that the literature provides parameters that allow reaching full accuracy recovery for different models and datasets. For instance, for quantization, we track the error of *4-bit quantization*, known to recover for every model (Alistarh et al., 2017; Markov et al., 2022). For sparsification, (Lin et al., 2017; Renggli et al., 2019) as well as for matrix decomposition (Vogels et al., 2019), we use different reference parameters according to their baselines. For details, please refer to Tables 2 and 3. An interesting consequence of this choice is that, since we guarantee ℓ_2 compression error, which is a small constant factor of the error of these theoretically-justified approaches, we inherit similar convergence guarantees, as per Nadiradze et al. (2021).

4 The L-GreCo Framework

Overview. We now describe a general algorithm to solve the constrained optimization problem from the previous section. Our algorithm makes layer-wise decisions in order to balance the magnitude of the compression error and the compressed size of the model. As inputs, our algorithm takes in the uncompressed layer sizes $size(\ell, \perp)$, a set $\mathcal{G}$ of accumulated gradients per layer (which will be used to examine compression error), as well as a fixed error bound $\mathcal{E}_{\max}$. Specifically, at a given decision step, the objective is to find an optimal mapping of each layer ℓ to a compression level c_ℓ, such that the norm of the total compression error, computed over the set of accumulated gradients $\mathcal{G}$ does not surpass $\mathcal{E}_{\max}$, but the total compressed size of the model $\sum_{\ell=1}^{L} size(\ell, c_\ell)$ is minimal for this error bound.

This formulation is reminiscent of the *knapsack problem*: the error is the size of the knapsack, and the compressed size is the value we wish to optimize. In this formulation, the problem would have an efficient optimal algorithm using dynamic programming (DP). However, the squared L_2 error is not discrete, so we cannot directly apply this approach. Instead, we reduce this to a solvable problem by *discretizing* the possible set of error values. Since errors are monotonic and we can use a very fine discretization without significant efficiency loss, it is unlikely that we would miss the optimal solution by a significant amount. For illustration, in our implementation, we use $D = 10000$ as a discretization factor (i.e. steps of size $\mathcal{E}_{\max}/D$).

The Algorithm. The procedure, presented in Algorithm 1, works as follows. First, we compute the data needed for the algorithm for all layers and all considered compression parameters (lines 1-10), corresponding to errors and compressions for each possible choice. Then, we execute a dynamic programming algorithm to solve the following problem. We want to compute the minimum total size given total compression error E in the first ℓ layers $compressedsize(\ell, E) = \min_\ell compressedsize(\ell - 1, E - error(\ell, c_\ell)) + size(\ell, c_\ell)$. To achieve this, for each layer we want to consider, we run over all error increments and all possible compression parameters and minimize the total compressed size for the current total compression error (lines 12-22), saving the compression parameter with which we obtain the minimum. Then, in lines 23-27, we find the error increment achiev-

Algorithm 1 L-GreCo adaptive compression

Input: Model Layers L_i, accumulated gradients G_i, compression parameters $C = \{c^1, c^2, \ldots, c^k\}$, static default compression parameters to improve C_i^d, discretization factor D
Output: Compression assignments $c_\ell \in C$ for each layer ℓ

N = number of layers
Compute $\mathcal{E}_{\max}$ for the default compression parameters C_i^d
Compute discretization step $\mathcal{E}_{\max}/D$.
Costs matrix $N \times |C|$ where position i, j has a value of the size of layer i compressed with compression parameter c^j.
Errors matrix $N \times |C|$ where position i, j has a value of the discretized L_2 of the compression error when the accumulated gradients of layer i are compressed with parameter c^j.
DP matrix $N \times (D+1)$ filled with ∞ values.
PD matrix $N \times (D+1)$.
// Initialization of the cost tables:
for $c \in C$ **do**
    $DP[1][Errors[1][c]] = Costs[1][c]$
    $PD[1][Errors[1][c]] = c$
end for
// Dynamic programming algorithm
for Layer $l_i := 2..N$ **do**
    for $c_i \in C$ **do**
        for $e_i := Errors[l_i][c_i]..D$ **do**
            $t = DP[l_i - 1][e_i - Errors[l_i][c_i]] + Costs[l_i][c_i]$
            if $t < DP[l_i][e_i]$ **then**
                $DP[l_i][e_i] = t$
                $PD[l_i][e_i] = c_i$
            end if
        end for
    end for
end for
$err_{min} = argmin(DP[N])$
// Reconstruction of the optimal parameters
for $l_i = N..1$ **do**
    $result[l_i] = PD[l_i][err_{min}]$
    $err_{min} = err_{min} - Errors[l_i][result[l_i]]$
end for
return $result$

ing the lowest total compressed size and reconstruct the compression parameter mapping—obtaining the result.

System Implementation. We integrate `L-GreCo` between the user training code and the communication system responsible for gradient compression and synchronization. We run the above algorithm periodically, e.g., once per training epoch, on a single designated worker; unless otherwise stated, this worker performs all steps. In between runs of the algorithm, we accumulate per-layer gradients in auxiliary buffers. We then build an L_2 error table for each layer, for every compression parameter in the user-provided range, and for the reference compression parameters set. To find the error, we simulate the compression/decompression of each layer with the given compression parameter without applying error feedback and compute the L_2 distance between the original and recovered vectors. Then, we run the DP algorithm. This provides us with the optimal compression mapping, which the designated worker broadcasts to the other workers. Then, on each worker, we save the parameters mapping in the communication engine.

Computational and memory costs. The algorithm assumes that we accumulate gradients in additional buffers, occupying the model size memory. The DP algorithm has $O(D|L||C|)$ time complexity and $O(|L|D)$ memory complexity. The actual timings for the algorithm are presented in Table 1. The overheads consist of 1. *error computation*, and 2. *running dynamic programming*. Dynamic programming takes a minor fraction of training time, whereas most of the overhead is caused by computation. However, both overheads are negligible compared to the speedups provided by `L-GreCo` (see Figures 3 and 11).

Table 1. Timing overheads for `L-GreCo` in relation to the total training time. Numbers in brackets represent error computation.

Model	Description	ResNet50
PowerSGD	0.56%[0.49%]	0.15%[0.14%]
QSGD	0.14%[0.13%]	0.04%[0.03%]
TopK	0.38%[0.35%]	0.33%[0.30%]

Communication details. In a data-parallel implementation, gradients become available right after the backward propagation of the corresponding layer, and are grouped into several buffers—called *buckets* in `Pytorch`)—allowing the overlapping of gradient communication with further computation. Thus, the communication of the first buckets is completely "hidden" by computation, whereas the synchronization of the last bucket becomes a significant part of the timing delay between steps. Thus, the transmission time of different buckets has a different impact on the training speed. Hence, in theory, optimizing for the compression ratio might yield suboptimal results in practice.

Optimizing for Time. To showcase the flexibility of L-GreCo, we augmented our system to allow us to measure the *actual synchronization time* for each gradient bucket, allowing the algorithm to optimize directly for *communication times*. We also reformulated the optimization problem: to minimize the *length of time* between the start of the first bucket synchronization and the end of the last one, rather than the compression ratio. Then, we train a regression model to learn the relation between the transmitted bucket sizes and gradient synchronization time. Thus, we obtained per-bucket coefficients $T(b)$, which we can apply for each layer in a respective bucket. Then we change the objective in the optimization problem(see Formula. 3) to:

$$\text{minimize} \sum_{\ell=1}^{L} size(\ell, c^\ell) * T(\ell) \text{ s.t. } \sum_{\ell=1}^{L} error(\ell, c^\ell) \leq \mathcal{E}_{\max}.$$

5 Experimental Validation

We experimentally evaluate L-GreCo across all existing compression strategies: quantization using QSGD, TopK sparsification, and low-rank approximation via PowerSGD.

5.1 Experimental setup

Infrastructure. Our evaluation uses commodity workstations with 4 or 8 NVIDIA RTX3090 GPUs. In the multi-node setting, we use 4 cloud instances with 4xRTX3090 GPUs provided by Genesis Cloud. Bandwidth measurements show that inter-GPU bandwidth values lie between 13 to 16 Gbps, and inter-node bandwidth in the cloud is up to 10 Gbps. We used Pytorch 1.10, openmpi/4.1.4, CUDA 11.3, NCCL 2.8.4, and cudnn/8.1.1.

Implementation. We implemented L-GreCo in PyTorch using `torch.distributed` hooks for PowerSGD and leveraging the open-source CGX framework (Markov et al., 2022) for basic quantization and sparsification operations.

Datasets and models. We examine two different DNN learning tasks: 1) image classification on the CIFAR100 (Krizhevsky, 2009) and ImageNet (Deng et al., 2009) datasets, and 2) language modeling on WikiText-103 (Merity et al., 2016). We used state-of-the-art model implementations and parameters provided by the PyTorch version of the NVIDIA Training Examples benchmark (Nvidia, 2020) and the `fairseq` library PyTorch examples (Ott et al., 2019). We used ResNet-18 for CIFAR-100 training with batch size 256, ResNet-50 in the mixed-precision regime for ImageNet with batch size 2048, and Transformer-XL and Transformer-LM trained in full-precision for WikiText-103, with batch sizes 256 and 2048, respectively. **All our experiments use the original uncompressed training recipes, without any additional hyperparameter tuning to account for gradient-compressed training.**

Baselines. The first natural baseline is uncompressed training, which sets our accuracy baseline. Matching MLPerf (Mattson et al., 2020), we set our accuracy threshold to 1% relative to uncompressed training. The second natural baseline is the *best existing manually-generated* gradient compression recipes. By and large, existing methods propose *uniform* per-layer compression to a given threshold, e.g. (Alistarh et al., 2017; Wen et al., 2017; Renggli et al., 2019; Vogels et al., 2019). For such baselines, we want to improve compressed size and training speed, possibly also improving final model accuracy. We found that the best choice of compression parameters for uniform per-layer assignment depends on the compression method, dataset, and task. For some experiments, we had to tune the uniform compression parameters to match baseline (non-compressed) results (see Tables 2 and 3).

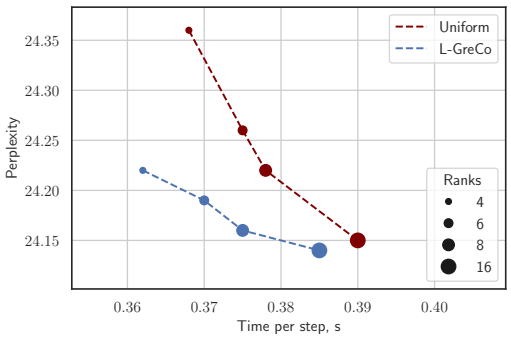


Figure 2. Perplexity (lower is better) vs. time per step (smaller is better) for different ranks of PowerSGD compression for the Transformer-XL wikitext-103 task with uniform or L-GreCo suggested compression schemes. Single node, 8 RTX3090 GPUs.

Parameter ranges. L-GreCo requires a range of possible compression parameters as input. We have always chosen this range to include the default compression parameters used in the literature. Moreover, we left a gap between the default parameter and the highest possible compression parameter (the right range bound) — otherwise, we are limited to matching L_2 error, and a gap between the default parameter and the lowest possible compression (the left range bound)—otherwise, we will not improve compression. We use the following approach for deciding ranges. Assume a default uniform compression parameter D, e.g., 4 quantization bits. For quantization and low-rank methods, the search space was defined as $[D/2, 2 * D]$ with an incremental step of 1. For sparsification, we chose $[D/10, 10 * D]$ with an increment of $D/10$.

The other two input parameters of L-GreCo are how frequently the algorithm is run and the warm-up period after which the compression is turned on. The first parameter matches the evaluation period (typically, 1 epoch). As we will see (Figure 6a), the compression ratio of the schemes returned by L-GreCo is relatively stable, so it does not need a frequent re-adjustment. The warm-up period equals the default learning rate warm-up period.

5.2 Evaluation results

Accuracy recovery. We first examine model accuracies using standard recipes for end-to-end training. For each experiment, we performed 3 runs with different seeds. We compare L-GreCo recovery with the uncompressed (baseline) and the best uniform per-layer compression parameters (uniform). The results are presented in Tables 2 and 3, including seed variability. The *compression ratio* represents actual transmission cost savings versus the uncompressed baseline. For each compression method and training task, we show the parameter value that provides the highest com-

Table 2. Accuracy recovery and compression ratios for different compression methods with uniform and adaptive schemes on image classification tasks. The compression ratios measure actual transmission savings. Values in brackets for L-GreCo compression ratios stand for improvements relative to the corresponding uniform compression.

Compression approach	Parameter choice	ResNet18 on CIFAR-100			ResNet50 on ImageNet		
		Default param	Accuracy	Compression ratio	Default param	Accuracy	Compression ratio
Baseline	N/A	-	76.60 ± 0.40	1.0	-	76.88 ± 0.16	1.0
QSGD	uniform	4 bit	76.80 ± 0.40	7.8	4 bit	77.38 ± 0.10	7.7
	L-Greco		76.46 ± 0.21	8.6 [1.10×]		76.77 ± 0.25	11.0 [1.41×]
TopK	uniform	1%	75.73 ± 0.46	48.1	1%	76.85 ± 0.06	45.6
	L-Greco		75.66 ± 0.35	182.0 [3.78×]		77.04 ± 0.27	122.0 [2.67×]
PowerSGD	uniform	rank 4	76.36 ± 0.28	72.2	rank 4	76.50 ± 0.37	66.5
	L-Greco		76.43 ± 0.37	133.9 [1.85×]		76.33 ± 0.27	96.2 [1.44×]

Table 3. Accuracy recovery and compression ratios for different compression methods with uniform and adaptive schemes on language modeling tasks. The compression ratios measure actual transmission savings. Values in brackets for L-GreCo compression ratios stand for improvements relative to the corresponding uniform compression.

Compression approach	Compression parameters	TransformerXL on WIKITEXT-103			TransformerLM on WIKITEXT-103		
		Default param	Perplexity	Compression ratio	Default param	Perplexity	Compression ratio
Baseline	N/A	-	23.82 ± 0.10	1.0	-	29.34 ± 0.12	1.0
QSGD	uniform	4 bit	23.82 ± 0.1	7.8	4 bit	29.39 ± 0.10	7.8
	L-Greco		24.11 ± 0.09	9.1 [1.16×]		30.03 ± 0.16	9.9 [1.26×]
TopK	uniform	10%	24.13 ± 0.14	4.9	10%	29.29 ± 0.09	4.9
	L-Greco		24.19 ± 0.13	12.8 [2.61×]		29.08 ± 0.20	25.6 [5.2×]
PowerSGD	uniform	rank 32	24.08 ± 0.12	14.0	rank 32	29.98 ± 0.09	15.0
	L-Greco		24.09 ± 0.15	20.8 [1.48×]		30.19 ± 0.09	26.5 [1.76×]

pression ratio while recovering final accuracy, i.e., further uniform compression leads to *worse* convergence.

Overall, results show that L-GreCo stays within the accuracy recovery limit of 1% multiplicative error (Mattson et al., 2020) for most tasks, often being close to the uniform baseline while consistently increasing the compression ratio across all the tasks and compression techniques. We stress that we did not perform task-specific parameter tuning. The gains are remarkably high for sparsification and low-rank techniques, where the search space and savings potential are higher. For instance, for Transformer-LM, we obtain **up to 5x higher compression** relative to the uniform baseline, with negligible accuracy impact. At the same time, L-GreCo induces $> 1\%$ multiplicative loss on quantization and low-rank compression for the highly-sensitive Transformer-LM model.[1] This is because our default compression range is too aggressive in this case; this can be easily addressed by adjusting the range—we chose not to do it for consistency.

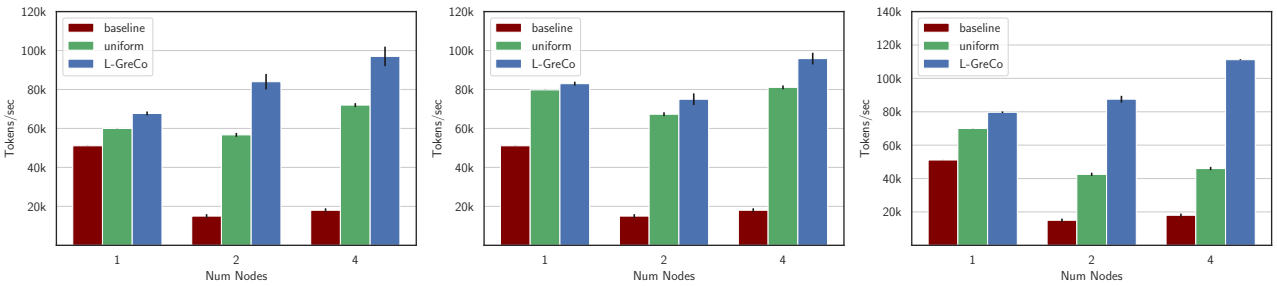
Further, we varied the uniform default parameters, specifically throttling the target PowerSGD rank for the Transformer-XL/WikiText-103 task. Figure 2 shows that L-GreCo provides a markedly better trade-off than uniform compression. We can see that L-GreCo provides better perplexity recovery while improving training speed.

Speedup results. For end-to-end training speedup improvements, we compare against standard uncompressed training ResNets and Transformers. We consider *weak scaling*, i.e., increase the global batch size while increasing the node count. (Performance improvements are higher for strong scaling.) We begin by examining training throughput results for *multi-node* training of Transformer-XL in Figure 3, executed in the cloud environment. (See Appendix Figure 11 for ResNet50 experiments.) This setting encounters a bandwidth bottleneck even at a lower node count, which is apparent given the poor performance of the uncompressed baseline. Tuned uniform compression partly removes this bottleneck: for instance, uniform PowerSGD/ResNet50 reaches 75% of ideal scaling on 4 nodes.

It is, therefore, surprising that automatic non-uniform compression can still provide significant improvements in this setting: relative to uniform compression, L-GreCo gives up to 2.4x speedup.

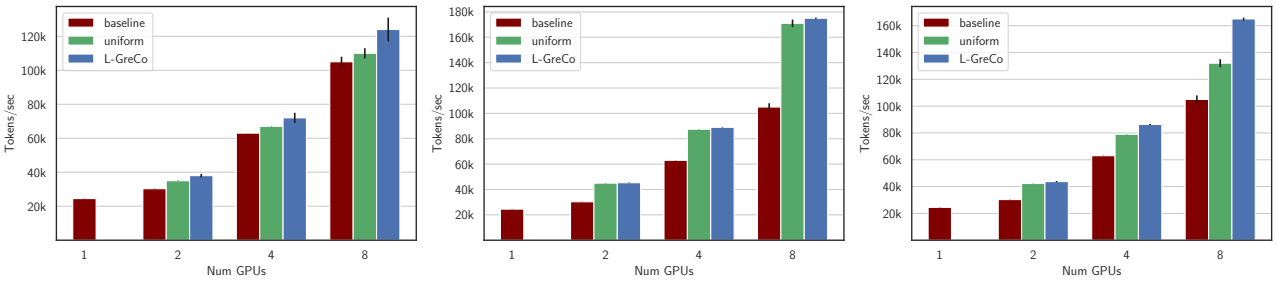
This suggests that non-uniform compression can be an effective strategy in this scenario, especially for layer-heterogeneous models such as Transformers.

[1] Specifically, our loss is of at most 0.85 perplexity relative to the uncompressed baseline. For this model, however, even basic FP16 training loses more than 1 point of perplexity vs FP32.



(a) PowerSGD (b) QSGD (c) TopK

Figure 3. Throughput for Transformer-XL (TXL) on WikiText-103. Multi-node, each node has 4 RTX3090 GPUs.



(a) PowerSGD (b) QSGD (c) TopK

Figure 4. Throughput for Transformer-XL (TXL) on WikiText-103. Single node, 8 RTX3090 GPUs.

We next examine results for single-node scaling from 1 to 8 GPUs, presented in Figure 4. For this model, using PowerSGD and TopK, `L-GreCo` leads to gains up to 25% *end-to-end* speedup compared to uniform, with negligible accuracy difference. For QSGD, the search space is very limited: uniform already uses 4 bits and provides very good scaling. Our adaptive method still provides 2% speedup compared to our well-tuned uniform compression and 50% speedup compared to non-compressed training, reaching $\geq$ 90% of ideal scaling. (Appendix Figure 10 presents ResNet50 results, which show lower improvements since training is weakly communication-bound in this setting.)

Overall, we note that `L-GreCo` provides statistically-significant performance improvements over static uniform compression (especially given heterogeneous models) when applied to all considered compression methods, with negligible impact on accuracy.

Profiling. In order to explore the compression overhead, we run the profiling of the training. The result is presented in Figure. 5. We compare operation timings for the original(uncompressed) training and training where the gradients are compressed with PowerSGD, rank 32. We can see that relatively expensive compression (PowerSGD is more time-consuming than QSGD and optimized TopK) takes less than 10% of the step time.

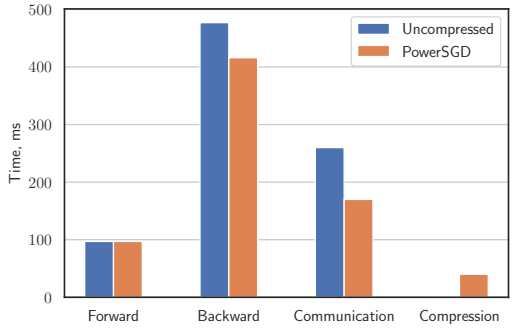
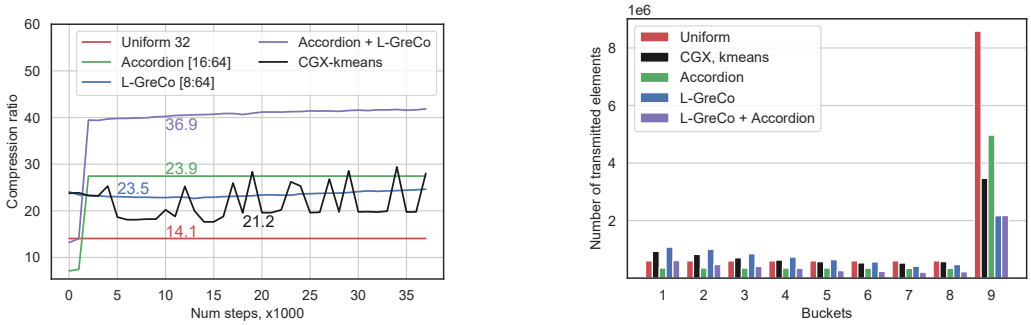


Figure 5. Profiling of the training without compression vs PowerSGD compression, rank 32. Transformer-XL model on WikiText-103 dataset. Single node, RTX3090 GPUs.

5.3 Comparison with other adaptive methods

So far, we have used uniform compression as our baseline. We now compare `L-GreCo` with prior works on adaptively choosing compression parameters. We consider the `Accordion` (Agarwal et al., 2021) and `CGX` (Markov et al., 2022) approaches, as they are the closest in terms of scope and the most general in terms of applicable compression methods. We perform our comparison on the Transformer-XL model on WikiText-103, as 1) it is a model that is sen-



(a) Compression ratio over time.

(b) Compression per layer group (bucket).

Figure 6. Adaptive compression using `L-GreCo` versus other methods, for PowerSGD compression on Transformer-XL. The left plot shows the dynamics of the compression ratio during training, marking the *average compression ratio*. The right plot presents the transmitted number of elements per bucket averaged over time. Buckets are in communication order.

sitive to gradient compression, 2) has high heterogeneity of layers, and 3) suffers from bandwidth bottlenecks (see Figure 3). We use PowerSGD as the compression method, as `Accordion` specifically optimizes for it. We run the experiments in two distributed settings: single node with 8 GPUs and multi-node, which includes 4 machines with 4 GPUs each. We compare compression and throughput (processed samples per second) for each method.

Here, since we aim to maximize speedup without dropping accuracy, we tune the compression parameters for each algorithm, the chosen compression method, and training task (without changing the training hyperparameters, e.g., learning rate, weight decay, etc.) so that we get the best timing results with the final model accuracy within the 1% MLPerf standard. The best range of parameters for `L-GreCo` turns is ranks $[8, 64]$ (see Table 4) with default rank 32.

Table 4. Comparison of `L-GreCo` with other adaptive algorithms on Transformer-XL using PowerSGD.

Adaptive algorithm	Param. Range	Ratio	Single node Tokens/s	Multi-node Tokens/s
Uniform	32	14.1	110k	72k
L-GreCo	8 - 64	23.5	**144k**	150k
Accordion	16, 64	23.9	114k	107k
CGX, kmeans	8 - 64	21.6	124k	112k
L-GreCo + Accordion	8 - 128	36.9	138k	**176k**

Global TopK Comparison. The optimization problem of minimizing gradient error magnitude given the desired compression ratio - can be alternatively solved by taking a global TopK in the case of gradient sparsification, which in this case minimizes total error. However, this method has several drawbacks. First, the global TopK requires careful fine-tuning and hyperparameter search in order to converge when low densities are used. `L-GreCo`, in turn, does not try to minimize the global compression error – it tries to match it to the compression error of the uniform *layer-wise* compression that recovers accuracy. Also, `L-GreCo` aims to maximize compression, meaning that each layer has a contribution to gradient synchronization. This may not be the case for global TopK: at high sparsities, some layers could have gradient zero for several steps, impacting model quality. The second disadvantage of global TopK is the actual speedup. As we discussed in Section 4, in modern data-parallel frameworks, gradients are synchronized in parallel with computation for the sake of efficiency, hiding communication costs behind computation. However, in global TopK, one has to wait until all layer gradients are produced, then perform compression and communication. In this case, the loss of performance due to non-overlapped communication may be higher than the improvements due to compression. To confirm this, we have implemented the algorithm using `torch.distributed` hooks and ran the RN18/CIFAR100 training. Even on this relatively small model, global TopK is 10% slower than `L-GreCo` when applied with a similar global density, in this case 0.25%.

CGX Comparison. The adaptive compression of `CGX` is based on `kmeans` and maps layers into a 2-dimensional space (layer size vs. L_2-error). The algorithm clusters layers into several groups and assigns predefined compression parameters to the layers in the groups. We have implemented this logic with PowerSGD compression. We used rank 32 as default, and the best (in terms of compression) range was from 8 to 64, using 6 layer clusters. The results are shown in the Table 4. In short, `L-GreCo` improves upon the kmeans approach by up to 33%. In Figure 6b, we observe that `L-GreCo` picks parameters such that the largest and the last bucket are compressed the most, whereas the `kmeans` algorithm chooses worse compression parameters for those layers in some iterations.

Rethink-GS Comparison. Sahu et al. (2021) suggest a sparcification method, which is technically adaptive hard-threshold sparsification changes the number of transmitted elements based on the gradient distribution. We have run the `L-GreCo` training of ResNet18/CIFAR-100 in the setup described in the paper. We used 1% density as a default parameter for `L-GreCo` and the search range was $[0.1\%, 10\%]$. For `Rethink-GS` we used parameter $\lambda = 4.72 \times 10^{-3}$.

As a result, `L-GreCo` improves upon `Rethink-GS` by 17% in terms of compression ratio ($6.7\times$ vs $5.7\times$) while also improving the final accuracy - 71.7% vs 71.4% (the numbers differ from the ones we show in Table. 2 as here we used the setup from (Sahu et al., 2021)). We note that our framework did not require *any hyperparameter tuning at all* for this experiment, whereas Rethink-GS requires careful tuning of the hard-threshold λ parameter.

Accordion Comparison. `Accordion` adapts compression by detecting critical regimes during training. It accepts two possible compression modes (corresponding to low and high compression) and has a threshold error parameter η. It collects the gradients and periodically decides the parameter to use based on gradient information for each layer. We have implemented Accordion using the `torch.distributed` hook, used for PowerSGD. For the parameter η, we chose the value of 0.5 suggested by the authors and tried to hand-tune the best pair of low and high compression parameters for each model, with which training converges to an accuracy that is within MLPerf bounds. We ran this algorithm on Transformer-XL/Wikitext-103, and found that the best pair of parameters (in terms of training time without losing accuracy) are high compression rank 16 and low compression rank 64.

In Figure 6, one can observe the dynamics of the average compression ratio over the training of `Accordion` relative to `L-GreCo`. We notice that `Accordion` chooses a low compression rank for almost all layers during the first period of training and a high compression rank for the rest of the training time, leading to completely bimodal uniform compression. This suggests that Accordion may not really exploit the heterogeneous nature of DNN models. Therefore, the optimizations of `Accordion` and `L-GreCo`, respectively, could be seen as *orthogonal*: `Accordion` focuses on varying the amount of *average compression* during training, whereas `L-GreCo` finds an optimal way of reaching this average level by setting layer-wise targets.

L-GreCo+ Accordion. With this in mind, we combined these two algorithms: We first executed `Accordion` to get the suggested parameters for each layer and used these parameters as the default set of parameters in `L-GreCo`, used to define the maximal error of the DP algorithm (see line 2 in Algorithm 1). Thus, `Accordion` determines the model sensitivity to gradient compression at different points in training, while `L-GreCo` finds the best mapping of compression parameters per layer. In Figure 6, we see that the resulting combination (`L-GreCo` with range $[8, 128]$ and Accordion with high=16, low=64) provides better compression ratios, without accuracy drop.

We also compare the performance of the two algorithms in isolation (see Table 4). We observe that, despite the fact that the theoretical compression ratio suggested by `Accordion` is essentially the same as that of `L-GreCo`, the `Accordion` throughput is less by around 30%. This is explained by the fact that `L-GreCo` compressed the last transmitted layers (buckets) to higher levels, leading to significantly-improved total transmission time. Specifically, in Figure 6b, we observe that `L-GreCo` transmits twice fewer elements in the last bucket relative to `Accordion`. Moreover, combining `L-GreCo` with `Accordion` improves the compression ratio by 50%, and training time by up to 66% compared to `Accordion`.

Overall, `L-GreCo` improved practical compression relative to prior techniques. Of note, the highest compression ratio is achieved by the hybrid `Accordion` + `L-GreCo` method, which leverages layer-wise insights in terms of both sensitivity and training dynamics.

5.4 Evaluating the loss-based accuracy metric

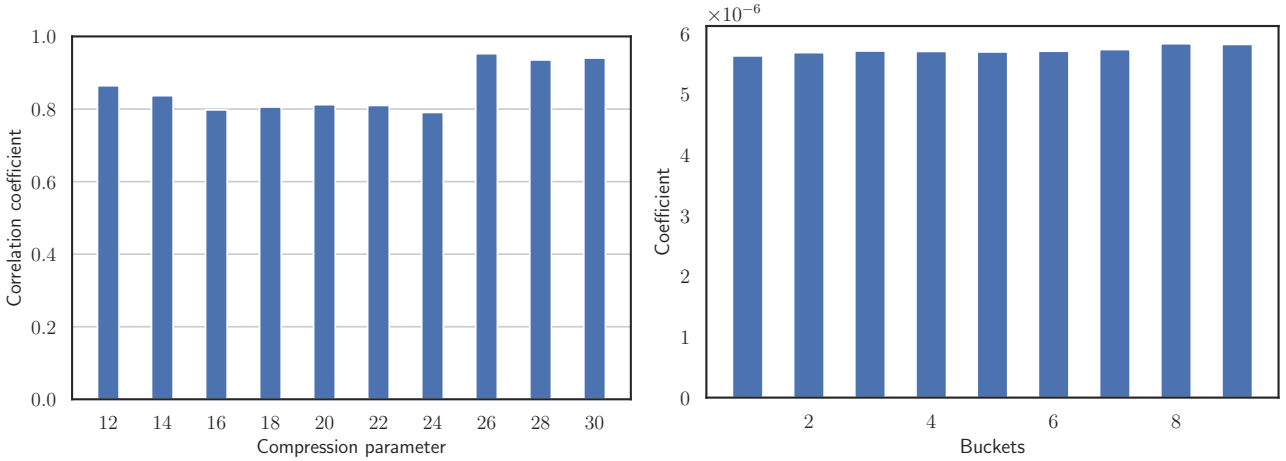
As discussed in Section 3, a key advantage of L-GreCo is that it can use any metric for measuring layer sensitivity to compression. To illustrate this, we investigated a loss-based metric, in which we collect model loss differences between uncompressed training and training with the gradient compression for certain layers, while other layers stay intact and use the loss difference as the sensitivity metric.

To compare the loss-based and error-based approaches, we evaluate the correlation coefficients of the metrics these two approaches provide. We observe (see Figure 7a) that the metric values from the two approaches have a high correlation and lead to very similar layer-wise compression parameters. Hence, since collecting loss-based metrics requires additional offline training runs, our usage of an error-based metric is justified.

5.5 Optimizing specifically for time

Considering that transmitted layer groups/communication buckets have different impacts on training performance, one may notice that compression may not always result in speedup. With this in mind, we have modified `L-GreCo` to *explicitly optimize the expected communication time*, rather than the compression ratio. See the last part of Section 4 for a detailed description of this algorithm variant.

We have run the resulting time-aware variant of `L-GreCo`



(a) Metrics correlation

(b) Coefficients

Figure 7. Correlation coefficients between metric values (a) for PowerSGD using loss-based and error-magnitude approaches as the sensitivity metrics. Timing coefficients per bucket (b) for timing-based approach. Transformer-XL/WikiText-103 model training with PowerSGD method.

with PowerSGD on Transformer-XL/WikiText-103 training. The linear model built on the timing data we collected (5000 samples - sets of compression ratios per bucket) has a score close to 1, meaning that we managed to predict the communication time using communicated bucket sizes almost perfectly. (The linear regression model was trained by running training for 50 steps with 5 steps of warmup for each set of compression parameters.)

We find that the per-bucket coefficients from the linear model are close to each other. Figure. 7b shows that the coefficients are uniform across the model. This means that each bucket's impact on communication time is proportional to the bucket size. Also, we noticed that the parameters we obtain with the modified algorithm are close to the parameters from the original `L-GreCo` algorithm. Given that, we figure that in the case of Transformer-XL the original L-GreCois close to optimal in terms of timing as well.

6 Conclusion

We proposed `L-GreCo`, an adaptive gradient compression algorithm that automatically identifies optimal layer-wise compression parameters, given a fixed error constraint. The `L-GreCo` algorithm finds the mapping of compression parameters such that 1) the total L_2 compression error matches a target known to recover accuracy, and 2) the total compressed size is minimal for this target.

Our approach is complemented by an in-depth exploration of the "correct" metrics, which capture the accuracy and performance impact of compression at the per-layer level. Specifically, we present evidence that minimizing local, per-layer compression errors leads to very similar results to minimizing global metrics such as output loss. Moreover, maximizing per-layer compression rates correlates very well with specifically minimizing total transmission time on existing data-parallel implementation.

We complemented these algorithmic and analytic contributions with extensive experimental validation across all families of gradient compression methods, showing that training with the layer-wise parameters suggested by `L-GreCo` recovers the baseline accuracy while the gradient compression ratio is substantially increased. `L-GreCo` improves training performance up to $2.5\times$, saving up to 5.2x communication compared to vanilla compression, and up to 122x relative to uncompressed training. Overall, our work provides a new approach for improving existing gradient compression methods at almost zero cost in terms of time and accuracy loss. Possible extensions of our method could consider *hybrid* strategies, which combine different families of compression techniques inside the same model.

REFERENCES

Achille, A., Rovere, M., and Soatto, S. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

Aflalo, Y., Noy, A., Lin, M., Friedman, I., and Zelnik, L. Knapsack pruning with inner distillation. *arXiv preprint arXiv:2002.08258*, 2020.

Agarwal, S., Wang, H., Lee, K., Venkataraman, S., and Papailiopoulos, D. Adaptive gradient communication via critical learning regime identification. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems*, volume 3, pp. 55–80, 2021.

Agarwal, S., Wang, H., Venkataraman, S., and Papailiopoulos, D. On the utility of gradient compression in distributed training systems. *Proceedings of Machine Learning and Systems*, 4:652–672, 2022.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720, 2017.

Chen, C. Y., Choi, J., Brand, D., Agrawal, A., Zhang, W., and Gopalakrishnan, K. Adacomp: Adaptive residual gradient compression for data-parallel distributed training. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, pp. 2827–2835, 2018.

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.

Dryden, N., Jacobs, S. A., Moon, T., and Van Essen, B. Communication quantization for data-parallel training of deep neural networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, pp. 1–8. IEEE Press, 2016.

Faghri, F., Tabrizian, I., Markov, I., Alistarh, D., Roy, D. M., and Ramezani-Kebrya, A. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

Frantar, E. and Alistarh, D. SPDY: Accurate pruning with speedup guarantees. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 6726–6743. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/frantar22a.html.

Karimireddy, S. P., Rebjock, Q., Stich, S., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pp. 3252–3261. PMLR, 2019.

Krizhevsky, A. Learning multiple layers of features from tiny images. pp. 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Lim, H., Andersen, D. G., and Kaminsky, M. 3lc: Lightweight and effective traffic compression for distributed machine learning. *arXiv preprint arXiv:1802.07389*, 2018.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Markov, I., Ramezanikebrya, H., and Alistarh, D. Cgx: Adaptive system support for communication-efficient deep learning, 2022. URL https://arxiv.org/abs/2111.08617.

Mattson, P., Reddi, V. J., Cheng, C., Coleman, C., Diamos, G., Kanter, D., Micikevicius, P., Patterson, D., Schmuelling, G., Tang, H., et al. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Nadiradze, G., Markov, I., Chatterjee, B., Kungurtsev, V., and Alistarh, D. Elastic consistency: A practical consistency model for distributed stochastic gradient descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9037–9045, 2021.

Nvidia. Nvidia deep learning examples for tensor cores, 2020. URL https://github.com/NVIDIA/DeepLearningExamples.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Ramezani-Kebrya, A., Faghri, F., Markov, I., Aksenov, V., Alistarh, D., and Roy, D. M. NUQSGD: Provably communication-efficient data-parallel sgd via nonuniform quantization. *Journal of Machine Learning Research*, 22 (114):1–43, 2021.

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*, 2019.

Sahu, A., Dutta, A., M. Abdelmoniem, A., Banerjee, T., Canini, M., and Kalnis, P. Rethinking gradient sparsification as total error minimization. In *Advances in Neural Information Processing Systems*, volume 34, pp. 8133–8146. Curran Associates, Inc., 2021.

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*. Citeseer, 2014.

Shen, M., Yin, H., Molchanov, P., Mao, L., Liu, J., and Alvarez, J. M. Structural pruning via latency-saliency knapsack. *arXiv preprint arXiv:2210.06659*, 2022.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vogels, T., Karinireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances In Neural Information Processing Systems 32 (Nips 2019)*, 32, 2019.

Wang, H., Sievert, S., Charles, Z., Liu, S., Wright, S., and Papailiopoulos, D. ATOMO: Communication-efficient learning via atomic sparsification. 2018.

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. *arXiv preprint arXiv:1705.07878*, 2017.

Wu, Y.-C., Liu, C.-T., Chen, B.-Y., and Chien, S.-Y. Constraint-aware importance estimation for global filter pruning under multiple resource constraints. In *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

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 *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*, pp. 561–572. IEEE, 2021.