Coordinating Distributed Example Orders for Provably Accelerated Training

A. Feder Cooper* **Wentao Guo*** **Khiem Pham***

Tiancheng Yuan **Charlie F. Ruan** **Yucheng Lu** **Christopher De Sa**

Cornell University
{afc78, wg247, dkp45, ty373, cfr54, yl2967, cmd353}@cornell.edu

Abstract

Recent research on online Gradient Balancing (GraB) has revealed that there exist permutation-based example orderings for SGD that are guaranteed to outperform random reshuffling (RR). Whereas RR arbitrarily permutes training examples, GraB leverages stale gradients from prior epochs to order examples — achieving a provably faster convergence rate than RR. However, GraB is limited by design: while it demonstrates an impressive ability to scale-up training on *centralized* data, it does not naturally extend to modern *distributed* ML workloads. We therefore propose *Coordinated Distributed GraB* (CD-GraB), which uses insights from prior work on kernel thinning to translate the benefits of provably faster permutation-based example ordering to distributed settings. With negligible overhead, CD-GraB exhibits a linear speedup in convergence rate over centralized GraB and outperforms distributed RR on a variety of benchmark tasks.

1 Introduction

Random reshuffling, which samples training-data examples without replacement, has become the *de facto* example-ordering method in modern deep-learning libraries [34], given that it tends to accelerate optimizer convergence in practice. However, some recent theoretical work has identified cases in which random reshuffling can lead to data orderings that have a poor effect on convergence [8, 36, 49]. This has encouraged a line of research to investigate if there exist provably better permutation-based orderings that afford greater scalability in training [22, 24, 31]. Notably, Lu et al. [24] connects permuted-order SGD to the *herding problem* [16], and proposes the herding-based online Gradient Balancing algorithm (GraB), which converges provably faster than random reshuffling, and does so with little memory or computational overhead. In fact, in follow-on work, Cha et al. [6] proves that GraB is optimal: in theory, GraB is the fastest possible permutation-based example ordering algorithm.

These results are very exciting, suggesting that GraB should unseat random reshuffling as the example ordering method-of-choice for SGD; however, they only hold with respect to a *single* machine. GraB is optimal in settings with *centralized* data, but does not naturally translate to problems of modern-ML scale, which demand that training workloads be distributed across *multiple parallel* workers that each only have access to a subset of the training data. This drawback raises an important question:

Can we simultaneously achieve the scalability benefits of distributed training and provably faster permutation-based example ordering for SGD — both in theory and in practice?

In this work, we show that it is indeed possible to attain these twin objectives. To do so, we suggest the online **C**oordinated **D**istributed **Gra**diant **B**alance algorithm (CD-GraB), which leverages insights

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

*Equal contribution

from kernel thinning to elevate the herding framework of centralized GraB (GraB) to the parallel setting. Felicitously, as a side effect, this choice of formulation brings about positive practical performance benefits (that can also improve the empirical behavior of centralized GraB). Using the exact same assumptions as the original GraB paper, **we show analytically that coordinating example orders across parallel workers leads a linear speedup in convergence rate**. For T epochs and m parallel workers, each with access to n examples, CD-GraB's convergence rate is $\tilde{O}((mnT)^{-2/3})$ on smooth, non-convex objectives and $\tilde{O}((mnT)^{-2})$ under the Polyak-Łojasiewicz (P.L.) condition.[2]

We run a series of experiments to verify these improvements in practice, implementing CD-GraB on a single node that distributes computation across multiple GPUs. We also run an ablation study in order to disentangle the benefits of parallelism from the positive side effects of using kernel thinning to formulate the CD-GraB algorithm. Similar to how centralized GraB demonstrates improved generalization over centralized random reshuffling (RR), we observe that CD-GraB exhibits improved generalization over distributed random reshuffling (D-RR). Altogether, the success of our work suggests a new distributed training paradigm to explore in future work, which we call the *Order Server* (Section 6). In summary, we:

- Propose the online **C**oordinated **D**istributed **Gra**dient **B**alancing (CD-GraB) algorithm, which enables provably accelerated training using SGD in the parallel setting (Section 3);
- Prove that the convergence rate for CD-GraB exhibits a linear speedup over GraB, using the exact same assumptions as the original GraB paper (Section 4);
- Produce extensive empirical validation of CD-GraB's improved scalability on a variety of tasks in deep learning and on large-scale logistic regression (Section 5).

2 Preliminaries and Related Work

In this section, we discuss the preliminaries and prior scholarship on permutation-based example ordering, with particular attention paid to the centralized online Gradient Balancing Algorithm (GraB) [24]. This lays the groundwork for how our coordinated, distributed GraB algorithm (Section 3) imparts the efficiency guarantees of GraB to the parallelized regime (Section 4).

Ordering data examples during training. Training a model can be formulated as minimizing a differentiable loss function $f:\mathbb{R}^d\to\mathbb{R}$ over N data examples. The goal of this minimization is to obtain the target model weights $\boldsymbol{w}^*=\operatorname{argmin}_{\boldsymbol{w}} f(\boldsymbol{w})$, where $f(\boldsymbol{w})=\frac{1}{N}\sum_{j=1}^{N} f(\boldsymbol{w};j)$, for which $f(\boldsymbol{w};j)$ denotes the loss incurred on the j-th example. A typical training process iteratively updates the model parameters $\boldsymbol{w}$ by scanning over the N data examples repeatedly, with t-th scan (or epoch) following

$$\boldsymbol{w}_t^{j+1}=\boldsymbol{w}_t^{j}-\alpha\nabla f(\boldsymbol{w}_t^{j};\pi_t(j)),\ \forall j\in[N], \tag{1}$$

where α denotes the learning rate, and $\pi_t:[N]\to[N]$ denotes a permutation ordering[3] adopted in the t-th epoch from which the examples are chosen to compute gradients, $\boldsymbol{w}_t^1$ denotes the initial model weights for the t-th epoch, and $\boldsymbol{w}_t^j$ denotes the model weights after $j-1$ gradient updates in the t-th epoch.[4]

The choice of ordering π can have a significant effect on optimizer performance. Two popular methods, which can demonstrate convergence speedups in practice, are 1) random reshuffling (RR) [46], for which the permutations are random and differ over epochs, and 2) Shuffle Once (SO) [4, 13], for which a random permutation is computed once and remains fixed for all epochs. Recht and Ré [37] conducted the first theoretical investigation of RR, while subsequent works like Yun et al. [49] and De Sa [8] have given counterexamples in which RR leads to orderings that have a poor effect on convergence. Altogether, many studies indicate that RR and SO only provide efficiency benefits under certain conditions [14, 15, 30].

These limitations of RR and SO have motivated research to identify permutations that outperform random ones. Rajput et al. [36] introduces an RR variant that achieves improved convergence for quadratics by reversing the ordering every other epoch. Other non-RR-based methods pick efficient orderings based on correlations between adjacently selected examples. In a recent line of work, Lu et al.

[2] In this paper, we use $\tilde{O}$ by convention to hide logarithmic factors in the problem parameters.

[3] While without-replacement orderings are most common in large-scale learning [5], ordering strategies need not be permutations, e.g., with-replacement sampling [23, 32, 39] or curriculum learning [12, 27, 41].

[4] Note that we write (1) in terms of per-example-j gradients.

[22] proves that faster convergence is possible for SGD when the averages of consecutive stochastic gradients converge faster to the full gradient. Based on this result, in follow-on work Lu et al. [24] proposes the centralized online Gradient Balancing algorithm (GraB), which outperforms RR, and upon which we base this work.

2.1 GraB: Optimal, online, permutation-based example ordering for centralized ML

GraB is a permutation-based example-ordering algorithm that identifies provably better-than-random orderings *in centralized, single-node settings* for SGD. GraB finds such orderings by leveraging information in stale stochastic gradients from previous epochs to guide ordering in the next epoch. More formally, for smooth, non-convex objectives, Lu et al. [24] proves that any permutation π^* that guarantees

$$\max_{k\in[N]}\left\|\sum_{j=1}^{k}\nabla f(\boldsymbol{w};\pi^*(j))-\nabla f(\boldsymbol{w})\right\|_\infty=\tilde{O}(1)\quad(\nabla f(\boldsymbol{w})\text{ is the average gradient}),\tag{2}$$

will yield a convergence rate of $\tilde{O}((NT)^{-2/3})$ (for epochs T) for SGD, which is superior to the $O(N^{-1/3}T^{-2/3})$ convergence rate of random reshuffling [30].

GraB's connection to herding and balancing. To find such a permutation π^*, Lu et al. [24] connect (2) to the *herding problem* and vector *balancing* [16, 45]. Understanding why GraB does not naturally extend to the distributed setting — and our main contributions (Sections 3 and 4) — requires some additional details on the fundamentals of herding:

Given N vectors[5] $\{\boldsymbol{z}_j\}_{j=1}^N$ ($\boldsymbol{z}_j\in\mathbb{R}^d$), $\|\boldsymbol{z}_j\|_2\le 1$ ($\forall j$), herding identifies a permutation π^* such that

$$\max_{k\in[N]}\left\|\sum_{j=1}^{k}\left(\boldsymbol{z}_{\pi^*(j)}-\bar{\boldsymbol{z}}\right)\right\|_\infty=\tilde{O}(1),\qquad\text{where }\bar{\boldsymbol{z}}=\tfrac{1}{N}\sum_{j=1}^{N}\boldsymbol{z}_j.\tag{3}$$

It is clear that (3) generalizes (2), which is a specific case of herding in an optimization setting.

Harvey and Samadi solve (3) with a method called *balancing* [16]. Balancing uses a *signed* version of the herding problem to optimize any given permutation π to reduce the bound in (3). That is, balancing formulates the signed herding problem

$$\max_{k\in[N]}\left\|\sum_{j=1}^{k}s_{\pi(j)}\left(\boldsymbol{z}_{\pi(j)}-\bar{\boldsymbol{z}}\right)\right\|_\infty,\quad\text{where }\{s_j\}_{j=1}^N\in\{+1,-1\}.\tag{4}$$

Given a group of such signs $\{s_j\}_{j=1}^N$ and an arbitrary permutation π, Harvey and Samadi prove that Algorithm 1 produces a new permutation π' such that

$$\max_{k\in[N]}\left\|\sum_{j=1}^{k}\left(\boldsymbol{z}_{\pi'(j)}-\bar{\boldsymbol{z}}\right)\right\|_\infty\le\tfrac{1}{2}\max_{k\in[N]}\left\|\sum_{j=1}^{k}s_{\pi(j)}\left(\boldsymbol{z}_{\pi(j)}-\bar{\boldsymbol{z}}\right)\right\|_\infty+\tfrac{1}{2}\max_{k\in[N]}\left\|\sum_{j=1}^{k}\left(\boldsymbol{z}_{\pi(j)}-\bar{\boldsymbol{z}}\right)\right\|_\infty.$$

This says that, with new permutation π', the objective of (3) now approaches the bound of (4). Importantly, recent advances show that it is quite cheap to find a group of signs, such that (4) is on the order of $\tilde{O}(1)$ (e.g., Alweiss et al. [2], in Algorithm 2). We are therefore able to call Algorithm 1 repeatedly, which will eventually obtain the π^* that solves the $\tilde{O}(1)$ herding objective in (3).

Algorithm 1 Reordering Vectors based on Balanced Signs [Harvey and Samadi [16]]

input: a group of signs $\{s_j\}_{j=1}^N$, initial order π
initialize: two order-sensitive lists $L_{\text{pos}}\leftarrow[]$, $L_{\text{neg}}\leftarrow[]$.
for $j=1\ldots N$ **do**
  $L_{\text{pos}}.\text{append}(\pi(j))$ **if** s_j is $+1$ **else** $L_{\text{neg}}.\text{append}(\pi(j))$.
end for
return: new order $\pi':=\text{concat}(L_{\text{pos}},\text{reverse}(L_{\text{neg}}))$.

GraB's application of herding to gradient balancing. Lu et al. [24] applies this framework of herding and balancing to develop GraB, i.e., to minimize (2). The main challenge for the success of this approach is to find the right gradients $\boldsymbol{z}_j$ in the optimization context of (2). Notably, the herding and balancing framework requires the vector mean $\bar{\boldsymbol{z}}$ in advance. To satisfy this requirement, GraB "centers" the gradient vectors using a *stale mean*. That is, GraB runs the herding algorithm on vectors that are defined as

$$\boldsymbol{z}_j=\nabla f(\boldsymbol{w}_t^j;\pi_t(j))-\tfrac{1}{N}\sum_{p=1}^{N}\nabla f(\boldsymbol{w}_{t-1}^p;\pi_{t-1}(p)),\tag{5}$$

where $\boldsymbol{w}_t^p$ denotes the model weights after $p-1$ updates in the t-th epoch, and π_t denotes the permutation adopted in the t-th epoch. Lu et al. [24] proves that this definition of $\boldsymbol{z}_j$ preserves the

[5] Herding does not have an optimization context. Here, N does *not* refer to the number of data examples used in training (1); rather, $N\in\mathbb{Z}^+$ describes the size of a set of arbitrary vectors. We slightly abuse notation because we execute the herding subroutine on exactly N gradients (Section 3), which happen to equal the number of N examples.

benefits of balancing with negligible noise or overhead. The only overhead comes from storing the running average of the gradients in epoch $t-1$ to "center" the gradients in the subsequent epoch t.

With this approach, Lu et al. [24] proves that GraB demonstrates more efficient convergence than RR for SGD. Better still, Cha et al. [6] demonstrates that GraB is in fact the *optimal* permutation-based ordering method for SGD: it is not possible to produce a permutation-based ordering in the centralized setting that achieves a faster convergence rate for SGD.

Despite GraB's clear benefits over RR, it assumes local access to all examples. This assumption does not hold for popular, modern, parallel settings (e.g., parameter server [20]), in which workers only have access to subsets of examples. No present work has attempted to investigate GraB's applicability to this setting. While some work has studied distributed RR (D-RR) [18, 26, 38, 48], it remains an open question if GraB's efficiency benefits for SGD can be conferred to the modern-scale, distributed-ML setup.

3 CD-GraB: A Provably Efficient Ordering Algorithm for Distributed Training

Our main contribution is to elevate GraB to the parallel regime, so that distributed training can enjoy the efficiency benefits of provably better example ordering. Based on the preliminaries, we can now explain why this is not a straightforward task: **While GraB achieves the optimal convergence rate for SGD on centralized data, it does not naturally translate to a distributed setting** (Section 3.1). Our key insights for resolving these problems are to reformulate the herding framework in Lu et al. [24] to work in parallel, and to leverage insights from kernel thinning [3, 10, 11] to derive the *online* PairBalance algorithm, which solves this parallelized herding objective (Section 3.2). Lastly, we present the full-stack CD-GraB algorithm that makes our solution work in practice (Section 3.3). The server implements online PairBalance, which coordinates gradient information from the distributed workers in training epoch t in order to determine a provably efficient example order for the next epoch $t+1$ (Section 4).

3.1 Issues with GraB in the distributed setting

To clarify the issues with distributing GraB, we first need to define the distributed training setup more precisely. We consider the standard data-parallel regime with m parallel workers, where each worker keeps a copy of the model weights $\boldsymbol{w} \in \mathbb{R}^d$ and maintains $n = N/m$ local examples.[6] As in many data-parallel training applications,[7] such as geo-distributed model training [47], we assume *the data examples cannot be shared or moved across workers*. More formally, this setup can be expressed as

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \left[f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} f^i(\boldsymbol{w}) \right] \quad \text{with} \quad f^i(\boldsymbol{w}) = \frac{1}{n}\sum_{j=1}^{n} f^i(\boldsymbol{w}; j), \tag{6}$$

where $f^i(\boldsymbol{w}; j) : \mathbb{R}^d \to \mathbb{R}$, $j \in [n]$, denotes the loss incurred on the j-th example on the i-th worker for model weights $\boldsymbol{w}$. We can now consider running (1) using this setup, for which each worker scans over their n local-data examples using (potentially) different permutations. We denote $\pi_{t,i} : [n] \to [n]$ as the permutation-based ordering adopted on the i-th worker in the t-th training epoch. Adjusting (1) to accommodate the setup in (6), the update to the model can be summarized as

$$\boldsymbol{w}_t^{j+1} = \boldsymbol{w}_t^j - \frac{\alpha}{m}\sum_{i=1}^{m} \nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j)), \ \ \forall j \in [n]. \tag{7}$$

That is, in epoch t, each worker i selects their respective, local j-th example according to $\{\pi_{t,i}\}_{i=1}^n$ in order to compute stochastic gradients (Appendix).

Following this setup, Algorithm 1 no longer guarantees the $\tilde{O}(1)$ bound to the herding problem (3), a bound that is valid only when *all* data examples can be permuted *freely* [16]. This constraint is fine for centralized GraB, but, in distributed training, workers only have access to a *subset* of examples. Distributed training requires that *worker-specific permutations only involve the examples in their respective local subsets*. Further, recall that GraB uses stale means to center gradients (5) in order to solve the herding objective. This, too, causes problems in distributed training. In practice, it is typical to employ larger learning rates α for greater scalability [40]; larger α increases the discrepancy between averaged gradients in adjacent epochs, which, in turn, would make GraB's use of stale means unreliable.

[6] Without loss of generality, we assume the N examples are divided evenly among the m workers and n is even.

[7] One such popular paradigm is federated learning [28, e.g.]. Federated learning typically involves highly imbalanced loads, heterogeneous data, partial user participation, and additional privacy-preserving mechanisms. These characteristics are orthogonal to what we consider here for example order. If we were to allow for such data organization, we would need to assume non-global communication per iteration or additional constraints on how global communication occurs. For CD-GraB, we focus on the regime of using parallelism to accelerate training.

3.2 Our efficient solution: parallel herding and pair balancing

To address the limitations presented in the prior section, which preclude the direct application of GraB to distributed training, we will need to **1) reformulate the herding problem to fit the parallel setting, and 2) redesign how to do gradient balancing**, such that it both solves our new herding formulation and allows for reliability with higher learning rates. We now present our solution to both these problems; we introduce the *parallel herding* problem and the online PairBalance subroutine that solves it.

Parallel Herding. To extend herding to the parallel setting, consider the following setup: There are m workers, which each have local access to n vectors. Let $\boldsymbol{z}_{i,j} \in \mathbb{R}^d$ denote the vector indexed by j on the i-th worker. Assuming $\|\boldsymbol{z}_{i,j}\|_2 \leq 1$ $(\forall i \in [m], \forall j \in [n])$, the goal of parallel herding is to find m permutations, $\pi_1, \pi_2, ..., \pi_m$ where $\pi_i : [n] \to [n]$ $(\forall i \in [m])$, so as to minimize:

$$\max_{k \in [n]} \left\| \sum_{j=1}^{k} \sum_{i=1}^{m} \left(\boldsymbol{z}_{i,\pi_i(j)} - \bar{\boldsymbol{z}} \right) \right\|_\infty, \quad \text{with} \quad \bar{\boldsymbol{z}} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{z}_{i,j}. \tag{8}$$

When directly comparing (8) with (3), it is clear that parallel herding differs in two notable ways from the original herding problem. First, each permutation $\pi_i : [n] \to [n]$ $(\forall i \in [m])$ only decides the ordering of the n vectors that are associated with worker i. Second, the prefix sum taken in the objective norm is accumulated over all the workers (the inner sum from $i = 1...m$). This formulation naturally captures the setting in a distributed environment: **workers need to decide permutations collaboratively, and the worker-specific vectors are processed simultaneously rather than sequentially**.

Given that this formulation fits the distributed setting, we next need to show that parallel herding does in fact address the limitations posed by centralized GraB: that it is possible recover the original $\tilde{O}(1)$ herding bound, and that we can solve the issue of unreliable stale gradients (Section 3.1). The solution that we present in the remainder of this section is a new vector balancing subroutine: online PairBalance. To give an intuition, as its name suggests, online PairBalance leverages insights from kernel thinning to *balance* vector differences over vector *pairs*. This also eliminates the need to perform vector centering, and thus solves the stale mean problem.

Using kernel thinning to solve parallel herding. We call our solution to the parallel herding objective (8) *pair balancing*, which we derive from key insights in *kernel thinning* [3, 10, 11]. In particular, Dwivedi and Mackey show that it is possible to solve the herding objective in $\tilde{O}(1)$ **by only examining differences on *pairs of examples*** [10]. They derive an algorithm that generalizes Alweiss et al. [2, subroutine in Algorithm 2], which solves herding in $\tilde{O}(1)$ (Section 2), and does so by operating only on vector-pair differences.[8] This comes with a very useful property: eliminating the requirement of knowing the maximum vector norm ahead of time and centering the vectors (i.e., making all the vectors sum to zero) in order to solve the herding problem. This is the key to solving the parallel herding objective (8) in $\tilde{O}(1)$, and elevating the benefits of GraB to a distributed setting.

Following Dwivedi and Mackey [10], we will balance over paired vectors, and will do so in an *online* fashion (Section 3.3). This eliminates GraB's requirement of using a stale mean to center gradient vectors (Section 2.1), but still minimizes the parallel herding objective to $\tilde{O}(1)$. We defer proving this result to Section 4, and first describe our concrete algorithm. Online PairBalance applies Algorithm 1 on the"flattened" and "paired" sequence of all of the workers' paired-difference gradients, i.e.,

$$\boldsymbol{y}_{n(k-1)+i} = \boldsymbol{z}_{i,2k-1} - \boldsymbol{z}_{i,2k}, \quad \forall k \in [\tfrac{n}{2}], \quad i = 1...m.$$

That is, we fit these ordered-paired differences $\{\boldsymbol{y}_i\}_{i=1}^{mn/2}$ into the herding and balancing framework (Algorithm 1): if sign s is associated with $\boldsymbol{y}_{n(k-1)+i}$, then $\boldsymbol{z}_{i,2k-1}$ and $\boldsymbol{z}_{i,2k}$ receive s and $-s$, respectively.

3.3 The full-stack CD-GraB algorithm

Having solved the parallel herding problem with pair balancing, we now demonstrate how to bring everything together in an optimization context to *coordinate distributed gradient balancing* for distributed training. That is, we can now introduce our full-stack CD-GraB algorithm, which trains models in a distributed setting (Section 3.1) while efficiently ordering the examples by using PairBalance (Section 3.2, Algorithm 2) in an online manner.

[8] Dwivedi and Mackey minimize the maximum mean discrepancy (MMD) between a selected coreset and an empirical distribution. They develop a new self-balancing Hilbert walk on differences of *pairs of examples* to select exactly half of the dataset points, and solve coreset selection by iteratively halving the input vector sequence into balanced coresets then selecting and refining a candidate coreset to minimize MMD with the input sequence.

Algorithm 2 PairBalance

▷ The inputs, outputs and subroutine for this algorithm are order-sensitive

input: current running sum $\boldsymbol{r}$, paired vectors $\boldsymbol{z}_1, \boldsymbol{z}_2$

compute: $s, \boldsymbol{r} \leftarrow \text{RandomizedBalance}(\boldsymbol{r}, \boldsymbol{z}_1 - \boldsymbol{z}_2)$
return: s (sign for $\boldsymbol{z}_1$),
$-s$ (sign for $\boldsymbol{z}_2$),
$\boldsymbol{r}$ (updated running sum)

▷ Adapted from Alweiss et al. [2]
define subroutine: $\text{RandomizedBalance}(\boldsymbol{r}, \boldsymbol{c})$
compute: $p \leftarrow \frac{1 - \langle \boldsymbol{r}, \boldsymbol{c} \rangle}{2}$
compute: $s \leftarrow +1$ with probability p;
$s \leftarrow -1$ with probability $1-p$
update: $\boldsymbol{r} \leftarrow \boldsymbol{r} + s\boldsymbol{c}$
return: $s, \boldsymbol{r}$

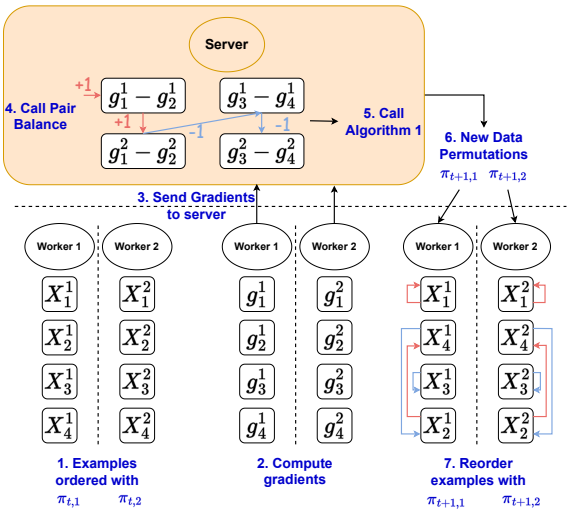


Figure 1: **Left:** The PairBalance algorithm, which the server runs online. **Right:** CD-GraB running on one server (top) and two workers (bottom). The workers do not share data examples.

We describe CD-GraB at two levels of abstraction: a high-level illustration (Figure 1, steps 1–7) and a detailed pair of worker-server algorithm statements (Figure 2). Since the workers only have access to a subset of the training data, in parallel they compute local, per-example stochastic gradients and send them to the server. The server simultaneously calls PairBalance online, which coordinates information from all the workers' gradients (i.e., using adjacent example-specific gradients) to determine the next epoch's worker-specific permutations. In more detail:

In epoch t, (Figure 1, step 1) the two workers have permutations $\pi_{t,1}$ and $\pi_{t,2}$, respectively. Each worker computes per-example gradients $\boldsymbol{g}_j^i$ (2; Algorithm 3:4), and sends them to the server (3; Algorithm 3:5). The server we implement functions as a parameter server [20]: It computes the average of the workers' per-example gradients (Algorithm 4:6), and sends it back to all workers (Algorithm 4:7) so that they can update their local models (Algorithm 3:6-7). Simultaneously, as the server receives gradients (Algorithm 4:5), it calls PairBalance (Algorithm 2) on adjacent vectors (4; Algorithm 4:4-13). PairBalance produces signs to supply to the reordering algorithm (Algorithm 1), which, using the current worker permutations $\pi_{t,i}$, produces the new per-worker permutations for the next epoch (5; Algorithm 4:14). In Figure 1, these correspond to $\pi_{t+1,1}$ and $\pi_{t+1,2}$, which the server then sends back to the respective workers (6; Algorithm 4:15). Lastly, before the start of the next epoch, the workers reorder their examples according to the new permutations (7; Algorithm 3:9).

4 Convergence Analysis

We next demonstrate formally that our CD-GraB algorithm (Section 3.3) confers the efficiency benefits of centralized GraB (Section 2.1) to the distributed setting. In brief, our main theoretical results show that **CD-GraB enjoys a linear speedup in convergence rate** under two sets of conditions: smoothness (Theorem 2) and the Polyak-Łojasiewicz (P.L.) condition (Theorem 3). **Both results guarantee that CD-GraB is faster than distributed random reshuffling (D-RR)**. Our proofs rely on Corollary 7 from Dwivedi and Mackey [10], which shows that, with high probability, RandomizedBalance (subroutine in Algorithm 2, from Alweiss et al. [2]) guarantees a $\tilde{O}(1)$ bound to the signed herding objective (4).[9]

To begin, we restate this result to cohere with our framework, for which the vectors $\boldsymbol{z}_j$ are gradients in an optimization context:

Theorem 1 (Corollary 7, Dwivedi and Mackey [10]). *Consider any vectors $\{\boldsymbol{z}_j\}_{j=1}^N$ ($\boldsymbol{z}_j \in \mathbb{R}^d$) with $\|\boldsymbol{z}_j\|_2 \leq 1$ supplied as input to the* RandomizedBalance *subroutine in Algorithm 2. Then for any $\delta > 0$, with probability at least $1-\delta$,* RandomizedBalance *outputs a sequence of signs $\{s_j\}_{j=1}^N \in \{-1, 1\}$ that satisfy $\max_{k \in [N]} \left\| \sum_{j=1}^k s_j \boldsymbol{z}_j \right\|_\infty \leq \tilde{A}$, where $\tilde{A} = \sqrt{2 \log\left(\frac{4d}{\delta}\right) \log\left(\frac{4N}{\delta}\right)} = \tilde{O}(1)$.*

[9] Corollary 7 from Dwivedi and Mackey [10] improves the result of Theorem 1.1 from Alweiss et al. [2].

Algorithm 3 CD-GraB Workers

require: m workers, $n \coloneqq \frac{N}{m}$ ex. per worker
input: initial $\boldsymbol{w}_1^1$, epochs T, learning rate α

receive: initial permutations $\leftarrow$ $\{\pi_{1,i}\}_{i=1}^m$
for epoch $t \coloneqq 1 \ldots T$ **do**
  $\triangleright$ Run in parallel for workers $i = 1 \ldots m$
  for example $j \coloneqq 1 \ldots n$ **do**
    compute: $\boldsymbol{g}_j^i \leftarrow \nabla f^i(\boldsymbol{w}_t^j, \pi_{t,i}(j))$
    send: $\boldsymbol{g}_j^i$ $\rightarrow$ j-th stochastic grad. $\boldsymbol{g}_j^i$
    receive: $\bar{\boldsymbol{g}}_j$ $\leftarrow$ avg. j-th stochastic grad. $\bar{\boldsymbol{g}}_j$
    update: $\boldsymbol{w}_t^{j+1} \leftarrow \boldsymbol{w}_t^j - \alpha \bar{\boldsymbol{g}}_j$
  end for
  receive: next permutation $\leftarrow$ $\pi_{t+1,i}$
  update: $\boldsymbol{w}_{t+1}^1 \coloneqq \boldsymbol{w}_t^{n+1}$
end for
return: $\boldsymbol{w}_{T+1} \coloneqq \boldsymbol{w}_{T+1}^1$

Algorithm 4 CD-GraB Parameter Server

require: m workers, $n \coloneqq \frac{N}{m}$ ex. per worker
input: epochs T

send: initial permutations $\{\pi_{1,i}\}_{i=1}^m$
for epoch $t \coloneqq 1 \ldots T$ **do**
  initialize: running sum $\boldsymbol{h} = \boldsymbol{0}$; empty list $\mathcal{S}$
  for example $j \coloneqq 1 \ldots n$ **do**
    receive: $\{\boldsymbol{g}_j^i\}_{i=1}^m$ from all workers i
    compute: avg. gradient: $\bar{\boldsymbol{g}}_j \leftarrow \frac{1}{m}\sum_{i=1}^m \boldsymbol{g}_j^i$
    send: $\bar{\boldsymbol{g}}_j$ to all the workers
    for worker $i \coloneqq 1 \ldots m$ **do**
      if $j \bmod 2 = 0$:
        $\boldsymbol{h}, s_{j-1}^i, s_j^i \leftarrow \mathsf{PairBalance}(\boldsymbol{h}, \boldsymbol{g}_{j-1}^i, \boldsymbol{g}_j^i)$
        $\mathcal{S}.\mathsf{append}(s_{j-1}^i)$; $\mathcal{S}.\mathsf{append}(s_j^i)$
      end for
    end for
  $\triangleright$ Call Alg. 1 for $i = 1 \ldots m$ on $\pi_{t,i}$ and $\mathcal{S}$
  compute: next permutations $\{\pi_{t+1,i}\}_{i=1}^m$
  send: $\{\pi_{t+1,i}\}_{i=1}^m$ to each worker i
end for

Figure 2: CD-GraB worker and server (here, a parameter server [20]) algorithms.

To integrate this result with our parallel setting, we need some additional assumptions that are standard in the literature on distributed optimization — that the variance of the per-example gradients on each worker is uniformly bounded (Assumption 1), and that the variance between worker-specific gradients is similarly bounded (Assumption 2). More precisely, following the distributed setup in (7), we denote the global loss gradient to be $\nabla f(\boldsymbol{w})$, each i-th worker's local loss gradient to be $\nabla f^i(\boldsymbol{w})$ ($\forall i \in [m]$), and each i-th worker's per-example loss gradients to be $\nabla f^i(\boldsymbol{w}; j)$ ($\forall j \in [n]$). We assume:

Assumption 1 (Bounded Gradient Variance). *For all $i \in [m]$ there exists a constant $\sigma > 0$ such that for all $j \in [n]$ and for all $\boldsymbol{w} \in \mathbb{R}^d$, it holds that $\left\|\nabla f^i(\boldsymbol{w}; j) - \nabla f^i(\boldsymbol{w})\right\|_2^2 \leq \sigma^2$.*

Assumption 2 (Bounded Data Heterogeneity). *There exists a constant $\varsigma > 0$ such that $\forall i \in [m]$, $\left\|\nabla f^i(\boldsymbol{w}) - \nabla f(\boldsymbol{w})\right\|_2^2 \leq \varsigma^2$.*

Lastly, we include one additional assumption from the original GraB paper [24]: we assume a cross norm $L_{2,\infty}$ (which can be easily adapted to L_2-smoothness by setting $L_{2,\infty}$ to be $\sqrt{d}L_2$).

Assumption 3 (Smoothness). *There exists constant $L_{2,\infty} > 0$ such that for any $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$, any $i \in [m]$, and any $j \in [n]$, it holds that $\left\|\nabla f^i(\boldsymbol{w}; j) - \nabla f^i(\boldsymbol{v}; j)\right\|_2 \leq L_{2,\infty}\|\boldsymbol{w} - \boldsymbol{v}\|_\infty$.*

Given these assumptions, we can prove a convergence guarantee for CD-GraB:

Theorem 2. *Suppose that Assumptions 1,2 and 3 hold. For any $\delta > 0$, if we set learning rate α to be*

$$\alpha = \min\left\{\frac{1}{16L_{2,\infty}(2n + \tilde{A}/m)}, \left(\frac{4F_1 m^2}{42L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2 nT + 18L_{2,\infty}^2 m^2 n^3 \sigma^2}\right)^{1/3}\right\},$$

where $F_1 = f(\boldsymbol{w}_1) - \inf_{\boldsymbol{w} \in \mathbb{R}^d} f(\boldsymbol{w})$ and $\tilde{A}$ comes from Theorem 1. Then, with probability at least $1 - T\delta$,

$$\frac{1}{T}\sum_{t=1}^T \|\nabla f(\boldsymbol{w}_t)\|_2^2 \leq \frac{9(F_1 L_{2,\infty}(\varsigma+\sigma)\tilde{A})^{2/3}}{(mnT)^{2/3}} + \frac{(72F_1 L_{2,\infty}\sigma)^{2/3} + 64F_1 L_{2,\infty}(2 + \tilde{A}/(mn))}{T}$$
$$= \tilde{O}\left(\frac{1}{(mnT)^{2/3}} + \frac{1}{T}\right).$$

We can also prove an accelerated rate for CD-GraB if we additionally assume the P.L. condition:

Assumption 4 (P.L. Condition). *We say the loss function f fulfills the P.L. condition if there exists $\mu > 0$ such that for any $\boldsymbol{w} \in \mathbb{R}^d$, $\frac{1}{2}\|\nabla f(\boldsymbol{w})\|_2^2 \geq \mu(f(\boldsymbol{w}) - \inf_{\boldsymbol{v}\in\mathbb{R}^d} f(\boldsymbol{v}))$.*

Theorem 3. *Suppose that Assumptions 1, 2, 3, and 4 hold. For any $\delta > 0$, we set constants $\tilde{W}$ and C_3 to be*

$$C_3 = \frac{(F_1 + \sigma^2/L_{2,\infty})\mu^2}{224 L_{2,\infty}^2 (\varsigma+\sigma)^2 \tilde{A}^2} \quad \text{and} \quad \tilde{W} = W_0(T^2 m^2 n^2 C_3),$$

where $\tilde{A}$ comes from Theorem 1, F_1 is from Theorem 2, and W_0 is the Lambert-W function. If we set learning rate $\alpha = \frac{2\tilde{W}}{Tn\mu}$ and if the number of epochs T satisfies

$$T \geq 10 + \frac{1}{\mu} 32 L_{2,\infty}(2 + \tilde{A}/(mn)) W_0((mnT)^2 C_3) = \tilde{O}(1),$$

then, with probability at least $1 - T\delta$, it holds that

$$F_{T+1} \leq \frac{1}{(mnT)^2}\left(\frac{(F_1 + L_{2,\infty}^2\sigma^2)\tilde{W}}{C_3} + \frac{112 L_{2,\infty}^2 (\varsigma+\sigma)^2 \tilde{A}^2 \tilde{W}^2}{\mu^3}\right) = \tilde{O}\left(\frac{1}{(mnT)^2}\right),$$

where $F_{T+1} = f(\boldsymbol{w}_{T+1}) - \inf_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$.

We prove Theorems 2 and 3 in the Appendix. Together, they show that CD-GraB exhibits a linear speedup in the number of workers m over GraB [24]'s convergence rates ($\tilde{O}((nT)^{-2/3})$ and $\tilde{O}((nT)^{-2})$, respectively).[10] under both smoothness and the P.L. condition. Further, CD-GraB's convergence rate of $\tilde{O}((mnT)^{-2})$ is faster than many previous rates,[11] such as the high probability bound of $\tilde{O}((mn)^{-1}T^{-2})$ for D-RR in Yun et al. [48].

5 CD-GraB in Practice: Distributed and Simulation Experiments

We next verify CD-GraB's accelerated convergence on a variety of empirical tasks.[12] For ease of comparison, we follow the experimental plan from the original GraB paper,[13] and add some additional large-scale logistic regression experiments. We also run an ablation study to isolate the effects of different improvements in CD-GraB. We do this because online PairBalance exhibits performance benefits that are separate from parallelism — namely, removing the need for gradient centering with a stale mean and allowing for higher learning rates (Section 3.2).[14]

Evaluating CD-GraB's convergence speedup. We use the following three tasks for evaluating distributed training efficiency: logistic regression on a large-scale mortgage application (New York 2017 subset, 244,107 examples with 18 features) [7] (Figure 3a), Long Short-Term Memory (LSTM) [17] on the WikiText-2 dataset [29] (Figure 3b), and autoregressive Multi-Layer Perceptron (MLP) on the M4 Weekly dataset [25] (Figure 3c). We measure the loss incurred on the entire training set (Full Train Loss) and task-appropriate test metrics during evaluation, with respect to both the number of epochs and wall-clock time. Regarding test metrics, we measure test accuracy for the mortgage application, perplexity for WikiText-2, and SMAPE for M4. Additional details regarding the datasets, models, and test metrics can be found in the Appendix.

For all three tasks, we use a single 128 GiB memory machine with 4 NVIDIA GeForce RTX 2080 Ti GPUs. For the mortgage application and WikiText-2 (Figures 3a and 3b), we launch $m = 4$ workers (processes), where each worker runs on one GPU. For the M4 task, we launch $m = 32$ workers, where each of the 4 GPUs hosts 8 process workers. We use NCCL as the distributed communication backend [33] for the mortgage application and WikiText-2 tasks, and GLOO [1] as the distributed communication backend for the M4 task.

As shown in Figure 3, we compare CD-GraB's convergence to the standard distributed-training example-ordering method: random reshuffling (D-RR). From all subfigures in Figure 3, we observe that CD-GraB outperforms the D-RR baseline significantly and consistently: CD-GraB exhibits better training loss and test metrics, measured against both the number of epochs and wall-clock time. We also note that the

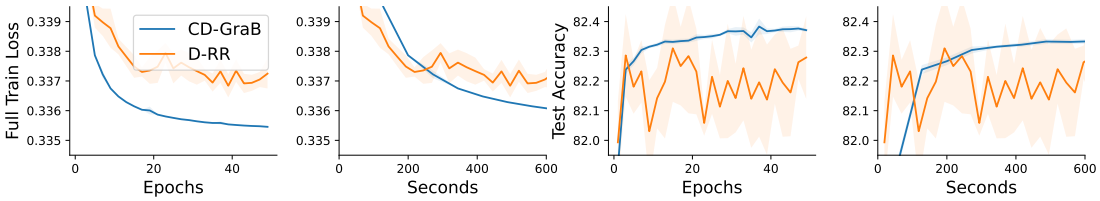
[10] For centralized GraB, the total number of examples $N = n$ and $m = 1$.

[11] These exclusively focus on the P.L. case, so we compare CD-GraB to them under the same condition.

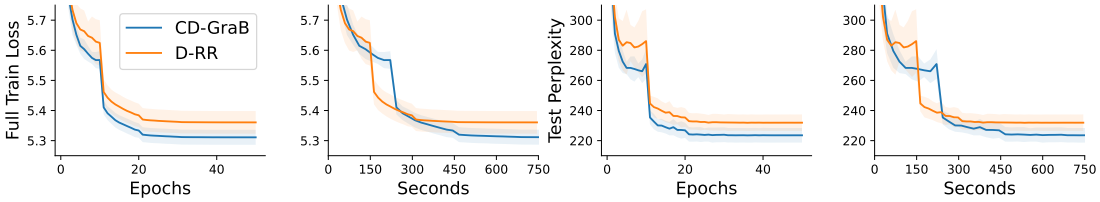
[12] Our GitHub repository is https://github.com/GarlGuo/CD-GraB.

[13] Following Lu et al. [24], for our LSTM experiment on WikiText-2, we set the embedding dimension to 32. We note that we can improve perplexity if we set the dimension higher.

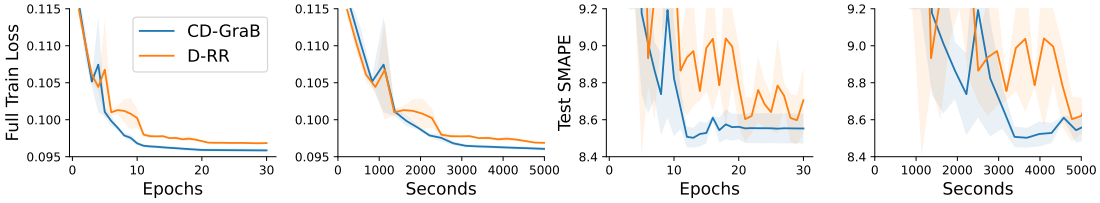
[14] GraB can also implement online PairBalance, in place of Balance [22] (Appendix).



(a) Logistic regression on mortgage application (NY 2017 subset) [7]



(b) LSTM [17] on WikiText-2 [29]



(c) Autoregressive MLP on M4 Weekly [25]

Figure 3: Convergence of CD-GraB in comparison to D-RR. For each experiment, we show train loss over epochs and time (**left** of each subfigure) and test performance over epochs and time (**right** of each subfigure). We run at least 3 random seeds, and plot the mean $\pm$ STD.

results for CD-GraB are much smoother than for D-RR. This is likely due to the variance of stochastic gradients during training, which CD-GraB reduces as a side-effect (so, too, does GraB, in comparison to RR). For smoother D-RR results, we can reduce the learning rate (Appendix). CD-GraB allows for the use of a larger learning rate, which accelerates training while preserving the final model's performance.

Ablation simulation study: the importance of coordination at large scale. CD-GraB has several design benefits over the original centralized GraB algorithm [24]: coordinating parallel workers' specific permutations using PairBalance on the server (Algorithm 2) and removing the dependency on a stale mean (Section 2.1), which enables the ability to using larger learning rates reliably (Section 3.2). Clearly, not all of these benefits come directly from distributing training. For example, being able to use larger learning rates, is a side effect of our solution to develop CD-GraB, not our main contribution. Therefore, we run a simulation ablation study to disentangle the relative importance of each of CD-GraB's efficiency benefits over GraB. To do so, we compare the convergence of CD-GraB to two additional baselines in the distributed setting, beyond D-RR: (1) **ID-GraB (Bal)**, where each independent worker runs GraB locally using RandomizedBalance (subroutine in Algorithm 2) to perform gradient vector balancing; (2) **ID-GraB (PairBal)**, where each independent worker runs GraB locally using PairBalance.

Figure 4 summarizes the results, with convergence curves for $m \in \{4,8,16,32,64\}$ workers training LeNet on CIFAR-10. We choose this task and architecture to cohere with the experiments done in the original GraB paper. For these experiments, we denote B to be the *aggregated* minibatch across all the workers, which refers to the number of stochastic examples used for an overall optimization step; each worker thus has a subset of this minibatch — an equivalently-sized subset of B examples.[15] We make two main observations. First, when scaling up training with more workers, CD-GraB converges increasingly faster than the no-coordination-ordering methods **ID-GraB (Bal)** and **ID-GraB (PairBal)**. This result aligns with our theory and intuition that, when the number of workers m increases, the parallel herding bound (8) will increase linearly if there is no coordination. Second, as we scale up to larger m, the convergence curves of **ID-GraB (Bal)** and **ID-GraB (PairBal)** gradually approach

[15] For example, if we have 4 workers with an aggregated minibatch size of 32, each worker would compute their respective local gradients with 8 examples, and then all-reduce these gradients to obtain the aggregated minibatch gradient for all 32 examples for the optimization step. We discard $N \bmod B$ examples at random to ensure n examples per worker.

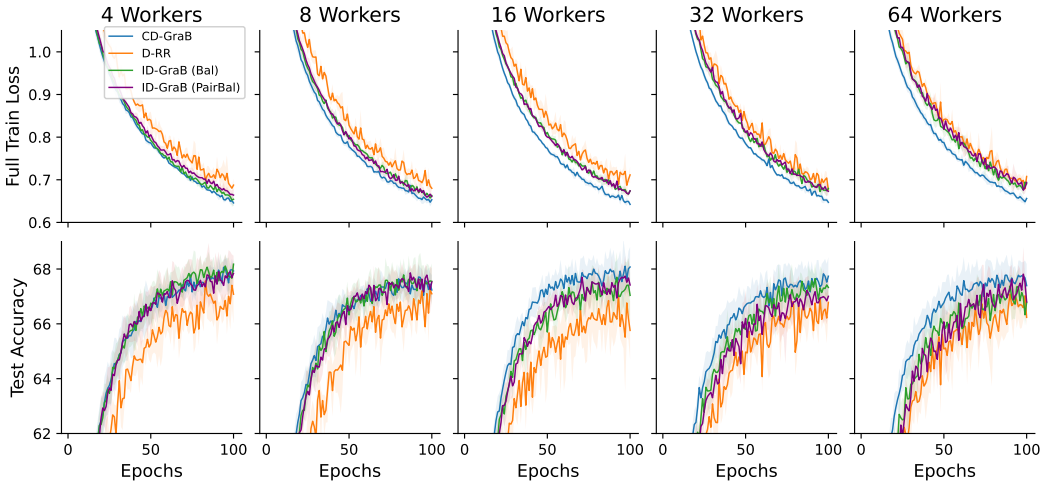


Figure 4: Convergence for CD-GraB, D-RR, ID-GraB (Bal), and ID-GraB (PairBal) training LeNet on CIFAR-10, with $m \in \{4,8,16,32,64\}$ workers. For each, the aggregated minibatch size per update is 64.

the curve for D-RR: at larger scales, herding-based example ordering will be no better than randomly permuting the dataset. Both observations give strong evidence that coordination (i.e., running online PairBalance on the server to coordinate per-worker permutations) is critical for accelerating training.

We note that all of these experiments use SGD, since both the theoretical results of the original GraB paper and our results for CD-GraB here are for SGD. In the Appendix, we additionally include results for training GPT-2 on WikiText-103, for which we use AdamW as the optimizer. We find that GraB with AdamW works in practice; however, our theory results do not directly apply to these experiments. We additionally include results on memory usage in the Appendix, which show that CD-GraB results in negligible overhead in practice.

6 Conclusion and Future Work: Toward an Order Server Architecture

We elevate the benefits of provably faster, permutation-based example ordering to the contemporary ML distributed-training setting. We focus on reformulating the online **Gra**dient **B**alancing algorithm (GraB) [24] because, even though it is the provably optimal permutation-based example-ordering method [6], it is limited by design to *centralized* settings (Section 3.1). To overcome these limitations, we redesign GraB's herding and balancing framework to account for parallel workers: A *parallel herding* objective, which we solve with an online PairBalance subroutine, based on key insights from kernel thinning [3, 10, 11]. PairBalance operates on ordered *pairs* of vectors to do *balancing*, which enables our full-stack, low-overhead, ***C****oordinated* and ***D****istributed* online CD-GraB algorithm. We give a full specification of our online CD-GraB algorithm (Section 3.3), provide convergence rate guarantees regarding its speedups on both 1) smooth non-convex and 2) P.L. objectives (Section 4), and verify these speedups in practice on single-node distributed tasks and a simulated ablation study (Section 5).

Both our theory and experiments demonstrate that CD-GraB really shines when there are multiple training epochs (Appendix). This is another reason that we do not emphasize experiments involving fine-tuning pre-trained models like GPT-2, as fine-tuning can be achieved in just a couple of epochs. As noted above, it is also more common to train such models using optimizers from the Adam family. In future work, we intend to extend the theory on GraB and CD-GraB to such optimizers, which would make the results on optimal, permutation-based example ordering more useful for base-model pre-training.

Pre-training from scratch would demonstrate the tremendous power of CD-GraB to scale to very large models; however, we did not have the training budget to perform such experiments for the present work. Further, to truly exercise the benefits of CD-GraB in such large-scale settings, future work should investigate moving beyond the single-node setup that we present. Notably, to train larger models, our results suggest a novel distributed training architecture. The ordering operation performed by the server (Algorithm 4) is *not* very latency sensitive; the server has the duration of the entire epoch t to compute the new permutations for the next, $t+1$ epoch. Given this relaxed latency requirement, and the success of our algorithmic results, it would be an exciting direction for future ML-systems research to invest in building an *Order Server* architecture. Such an architecture, which could be composed with traditional parameter servers, would afford the scalability benefits of CD-GraB to a host of massive-scale ML applications.

Acknowledgments

A. Feder Cooper is supported by Christopher De Sa's NSF CAREER grant. Yucheng Lu is supported by Meta Ph.D. Fellowship. We also acknowledge a gift from SambaNova Systems.

References

[1] Collective communications library with various primitives for multi-machine training, 2023. URL `https://github.com/facebookincubator/gloo`.

[2] Ryan Alweiss, Yang P Liu, and Mehtaab Sawhney. Discrepancy minimization via a self-balancing walk. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 14–20, 2021.

[3] Alessandro Barp, Carl-Johann Simon-Gabriel, Mark Girolami, and Lester Mackey. Targeted separation and convergence with kernel discrepancies. *arXiv preprint arXiv:2209.12835*, 2022.

[4] Dimitri P. Bertsekas. Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey. In *Optimization for Machine Learning*. The MIT Press, 2011.

[5] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[6] Jaeyoung Cha, Jaewook Lee, and Chulhee Yun. Tighter Lower Bounds for Shuffling SGD: Random Permutations and Beyond. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[7] A. Feder Cooper, Katherine Lee, Solon Barocas, Christopher De Sa, Siddhartha Sen, and Baobao Zhang. Is My Prediction Arbitrary? Measuring Self-Consistency in Fair Classification. *arXiv preprint arXiv:2301.11562*, 2023.

[8] Christopher De Sa. Random Reshuffling is Not Always Better. In *Advances in Neural Information Processing Systems*, 2020.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[10] Raaz Dwivedi and Lester Mackey. Kernel thinning. *arXiv preprint arXiv:2105.05842*, 2021.

[11] Raaz Dwivedi and Lester Mackey. Generalized Kernel Thinning. In *Tenth International Conference on Learning Representations*, 2022.

[12] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[13] Mert Gürbüzbalaban, Asuman E. Ozdaglar, and Pablo A. Parrilo. Convergence Rate of Incremental Gradient and Incremental Newton Methods. *SIAM Journal on Optimization*, 29 (4):2542–2565, 2019.

[14] Mert Gürbüzbalaban, Asu Ozdaglar, and Pablo A Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, 186(1):49–84, 2021.

[15] Jeff Z. HaoChen and Suvrit Sra. Random Shuffling Beats SGD after Finite Epochs. In *Proceedings of the International Conference on Machine Learning*, volume 97, pages 2624–2633, 2019.

[16] Nick Harvey and Samira Samadi. Near-Optimal Herding. In *Proceedings of The 27th Conference on Learning Theory*, volume 35, pages 1165–1182, 2014.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[18] Kun Huang, Xiao Li, Andre Milzarek, Shi Pu, and Junwen Qiu. Distributed Random Reshuffling over Networks. *arXiv preprint arXiv:2112.15287*, 2021.

[19] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

[20] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 583–598, USA, 2014. USENIX Association. ISBN 9781931971164.

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[22] Yucheng Lu, Si Yi Meng, and Christopher De Sa. A General Analysis of Example-Selection for Stochastic Gradient Descent. In *International Conference on Learning Representations*, 2021.

[23] Yucheng Lu, Youngsuk Park, Lifan Chen, Yuyang Wang, Christopher De Sa, and Dean Foster. Variance Reduced Training with Stratified Sampling for Forecasting Models. In *Proceedings of the International Conference on Machine Learning*, pages 7145–7155. PMLR, 2021.

[24] Yucheng Lu, Wentao Guo, and Christopher De Sa. GraB: Finding Provably Better Data Permutations than Random Reshuffling. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=nDemfqKHTpK`.

[25] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1): 54–74, 2020. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2019.04.014. URL `https://www.sciencedirect.com/science/article/pii/S0169207019301128`. M4 Competition.

[26] Grigory Malinovsky, Konstantin Mishchenko, and Peter Richtárik. Server-Side Stepsizes and Sampling Without Replacement Provably Help in Federated Optimization. *arXiv preprint arXiv:2201.11066*, 2022.

[27] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

[28] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[29] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. In *International Conference on Learning Representations*, 2018.

[30] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. Random Reshuffling: Simple Analysis with Vast Improvements. In *Advances in Neural Information Processing Systems*, 2020.

[31] Amirkeivan Mohtashami, Sebastian Stich, and Martin Jaggi. Characterizing & Finding Good Data Orderings for Fast Convergence of Sequential Gradient Methods. *arXiv preprint arXiv:2202.01838*, 2022.

[32] Deanna Needell, Rachel Ward, and Nathan Srebro. Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.

[33] NVIDIA. NVIDIA Collective Communication Library, 2023. URL `https://https://developer.nvidia.com/nccl`.

[34] PyTorch Contributors. DataLoader API, 2023. URL `https://pytorch.org/docs/stable/data.html`.

[35] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[36] Shashank Rajput, Kangwook Lee, and Dimitris Papailiopoulos. Permutation-Based SGD: Is Random Optimal? In *International Conference on Learning Representations*, 2022.

[37] Benjamin Recht and Christopher Ré. Toward a Noncommutative Arithmetic-geometric Mean Inequality: Conjectures, Case-studies, and Consequences. In *Conference on Learning Theory*, volume 23, pages 11.1–11.24, 2012.

[38] Abdurakhmon Sadiev, Grigory Malinovsky, Eduard Gorbunov, Igor Sokolov, Ahmed Khaled, Konstantin Burlachenko, and Peter Richtárik. Federated Optimization Algorithms with Random Reshuffling and Gradient Compression. *arXiv preprint arXiv:2206.07021*, 2022.

[39] Mark Schmidt, Nicolas Le Roux, and Francis R. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[40] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't Decay the Learning Rate, Increase the Batch Size. In *International Conference on Learning Representations*, 2018.

[41] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, pages 1–40, 2022.

[42] Evangelos Spiliotis, Andreas Kouloumos, Vassilios Assimakopoulos, and Spyros Makridakis. Are forecasting competitions data representative of the reality? *International Journal of Forecasting*, 36(1):37–53, 2020.

[43] Merity Stephen, Xiong Caiming, Bradbury James, and Richard Socher. Pointer sentinel mixture models. *Proceedings of ICLR*, 2017.

[44] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL `https://aclanthology.org/W18-5446`.

[45] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009.

[46] Bicheng Ying, Kun Yuan, Stefan Vlaski, and Ali H. Sayed. On the performance of random reshuffling in stochastic learning. In *2017 Information Theory and Applications Workshop (ITA)*, pages 1–5. IEEE, 2017.

[47] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35:25464–25477, 2022.

[48] Chulhee Yun, Shashank Rajput, and Suvrit Sra. Minibatch vs Local SGD with Shuffling: Tight Convergence Bounds and Beyond. In *International Conference on Learning Representations*, 2021.

[49] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Open Problem: Can Single-Shuffle SGD be Better than Reshuffling SGD and GD? In *Conference on Learning Theory*, 2021.

A Glossary

Term/Symbol	Explanation
GraB	Centralized online Gradient Balancing algorithm. We use this term to refer to the centralized algorithm developed in the original GraB paper (Lu et al. [24]).
CD-GraB	Coordinated and distributed online Gradient Balancing algorithm. We use this term to refer to the algorithm that constitutes our main contribution.
ID-GraB	Independent and distributed online gradient balancing. We implement this for our ablation study, to compare with coordinated and distributed GraB. There are two variants: One which uses the original GraB paper's online Balance algorithm (ID-GraB (Bal)), and one which implements our online PairBalance algorithm (ID-GraB (PairBal)).
RR	Random reshuffling algorithm. We use this to refer to its centralized variant.
D-RR	Distributed random reshuffling algorithm.
SO	Shuffle Once algorithm.
$\boldsymbol{x}$	Data-example vector; we do not use this in the math in the main paper, but do refer to examples in our schematic description for PairBalance ordering in Figure 1.
$\boldsymbol{z}$	Vector (for illustration under the herding context). For GraB and CD-GraB, these are gradients.
$\bar{\boldsymbol{z}}$	The average vector (for illustration under the herding context).
$\boldsymbol{z}_j$	The j-th component of a vector (for illustration under the herding context).
$\boldsymbol{z}_{i,j}$	The j-th component of the gradient on worker i (for illustration under our parallel herding framework).
$\boldsymbol{w}$	Parameters / model-weights vector.
f	Loss function.
$\nabla f(\boldsymbol{w})$	Global loss gradient.
$\nabla f^i(\boldsymbol{w})$	Local i-th worker's loss gradient.
$\nabla f^i(\boldsymbol{w};j)$	Local i-th worker's, j-th example's loss gradient.
π	A permutation; we study permutation-based example orderings.
T	Number of epochs.
t	Index for iterating over T epochs.
m	Number of workers (in this paper, workers are processes, potentially on different GPUs but on the same node). $m=1$ in the centralized setting.
i	Index for iterating over m workers .
n	Number of training-data examples per worker; equivalent to $\frac{N}{m}$.
N	Number of total training-data examples. $N=n$ in the centralized setting.
j	Index for iterating over examples.
$\boldsymbol{g}$	Gradient, taken with respect to the model weights $\boldsymbol{w}$ and data examples $\boldsymbol{x}$.
$\boldsymbol{g}_j^i$	Gradient associated with the j-th data example $\boldsymbol{x}$ on worker i.
s	A sign, either $+1$ or -1; related to the signed herding problem.
s_j^i	A sign, either $+1$ or -1, computed according to the j-th example gradient $\boldsymbol{g}_j^i$ for worker i; to be associated with the example $\boldsymbol{x}_j$ when determining a permutation ordering using Algorithm 1.

B Additional Details on the CD-GraB Algorithm and online PairBalance

In this Appendix, we provide more details on related work and our contributions. To start, we give a unified description of our online CD-GraB algorithm with prior work on herding, vector balancing, and kernel thinning (Appendix B.1), some more details on Alweiss et al. [2] that we elide in the main paper due to space constraints (Appendix B.2), conceptual details on implementing CD-GraB with a parameter server (Appendix B.3), and implementing our improved balancing algorithm (online PairBalance) in a centralized fashion to get additional improvements for GraB (Appendix B.4).

B.1 Distinguishing our contributions

We summarize our contributions in relation to prior work in a concise format. This kind of presentation would not be easily understandable without the appropriate background and context that we provide in the paper. This is why present it here, in the Appendix, so that (ideally) this is seen by the reader after finishing the main paper.

We emphasize that it is prior work that:

- Formulates the herding objective and solves it with vector balancing [16, 45] (Algorithm 1).
- Leverages ideas from herding and vector balancing (above) in an optimization setting to do permutation-based example ordering [24].
- Observes and proves that it is possible to solve the herding objective in $\tilde{O}(1)$ by only examining differences on pairs of examples (the overarching idea of PairBalance [10], which relies on the online RandomizedBalance subroutine [2]; see Algorithm 2).

Our contributions are to bring together all of this prior work in a novel way. We

- Translate the herding and balancing framework to the parallel setting via defining a parallel herding objective (8).
- Leverage prior work on herding in an optimization setting [24] so that we can do parallel herding in an optimization setting (Section 3).
- Execute *online* pair balancing on a server (Algorithm 2 on a running sum, Figure 1), i.e., do pair balancing in a streaming and asynchronous (rather than blocking) fashion from gradient vectors produced on distributed workers (Algorithm 2), on the flattened sequenced of paired-difference gradients (Section 3.2); this leads to an improvement over GraB, which relies on a stale mean.

B.2 More details on RandomizedBalance from Alweiss et al. [2]

In the subroutine for RandomizedBalance in Algorithm 2, we elide details about how the probability p is computed exactly as in Alweiss et al. [2]. We provide a more complete specification in Algorithm 5 written in terms of a single input vector (which, for us, is the vector containing the difference between adjacent gradients). Note that the difference here is in the use of a required parameter, constant upper bound w, which is used to compute the probability p. For clarity of presentation in the subroutine in Algorithm 2, we have set $w = 1$. Alweiss et al. [2] sets this threshold differently, which we still elide for simplicity.

Algorithm 5 Probabilistic Balancing with Logarithm Bound [Alweiss et al. [2]]

require: parameter w, used to compute probability
input: current running sum $\boldsymbol{r}$ vector, vector $\boldsymbol{z}_{\text{diff}}$
if $|\langle \boldsymbol{r}, \boldsymbol{z}_{\text{diff}} \rangle| > w$ or $\|\boldsymbol{r}\|_\infty > w$ **then**
    Fail
end if
compute: $p \leftarrow \frac{1}{2} - \frac{\langle \boldsymbol{r}, \boldsymbol{z}_{\text{diff}} \rangle}{2w}$
compute: $s \leftarrow +1$ with probability p;
    $s \leftarrow -1$ with probability $1-p$
update: $\boldsymbol{r} \leftarrow \boldsymbol{r} + s\boldsymbol{z}_{\text{diff}}$
return: s, $\boldsymbol{r}$

In practice, we actually do not use RandomizedBalance in our online PairBalance. We use the deterministic, greedy-ordering algorithm from the original Lu et al. [24, Algorithm 5] paper:

Algorithm 6 Balancing without normalization [Lu et al. [24]]

input: current running sum $\boldsymbol{r}$ vector, vector $\boldsymbol{z}_{\text{diff}}$
if $\|\boldsymbol{r}+\boldsymbol{z}_{\text{diff}}\| < \|\boldsymbol{r}-\boldsymbol{z}_{\text{diff}}\|$ **then** $s \leftarrow +1$ **else** $s \leftarrow -1$
update: $\boldsymbol{r} \leftarrow \boldsymbol{r} + s\boldsymbol{z}_{\text{diff}}$
return: $s, \boldsymbol{r}$

Note that, unlike Alweiss et al. [2] (Algorithm 5), Algorithm 6 from Lu et al. [24] cannot end up in a failure state.

In Alweiss et al. [2], Theorem 1.1 proves the $\tilde{O}(1)$ probabilistic bound for Algorithm 5 (See Theorem 1) for a restatement of this result in terms of our work). Corollary 7 of Dwivedi and Mackey [10] re-proves this result (which they mislabel as Alweiss et al. [2], Theorem 1.2, see Dwivedi and Mackey [10, Appendix R, p. 69]). They improve the constants and have a less conservative setting of the thresholds w. The proof is also very short and elegant, by relying on their Theorem 3.

B.3 Implementing CD-GraB with a parameter server

For our implementation of CD-GraB, we use a parameter server architecture [20]. For our purposes, this just entails computing the average gradient (used to update the model on all workers) on the server side. That is, the server (other than determining the ordering for the next epoch) also has the function of aggregating gradient information (in this case, a simple mean) to send back to the workers.

We have the server compute the average j-th gradient for illustrative purposes. We could, instead, implement the computation the average gradient as an all-reduce operation, in which each worker broadcasts their gradients to all other workers, so that they can each locally compute the average gradient to update their local models. We implement CD-GraB using a parameter server pattern to show that this is a plausible architecture to use with our coordinated and distributed example ordering algorithm. We could also implement a full parameter server system, for which the server also coordinates global model updates.

If we kept everything in our implementation the same and switched to all-reduce, then we would no longer be following a parameter server paradigm. In this case, the server would just function to determine example orders. It is this kind of paradigm that suggests the abstraction of an *order server*, which we mention briefly in Section 6: A server whose sole responsibility is coordinating worker information to determine example ordering.

In future work, we intend to explore a host of architectural possibilities — of building a full system that incorporates both traditional parameter server aspects with our new abstraction of an order server. For example, we could have parameter servers and order servers work in tandem in a distributed system to perform model training. To move beyond the single-node implementation we present in this paper, we intend to investigate the benefits and trade-offs associated with such design decisions in an actual implemented system.

B.4 Centralized online PairBalance

In Section 3.3, we provide a schematic diagram of how online PairBalance works for a distributed implementation using a parameter server (Figure 1). We also claim in Section 3 that online PairBalance can be applied to the original centralized GraB algorithm for improved empirical performance. We provide a schematic here, in Figure 5 (analogous to Figure 1), for online PairBalance for centralized GraB.

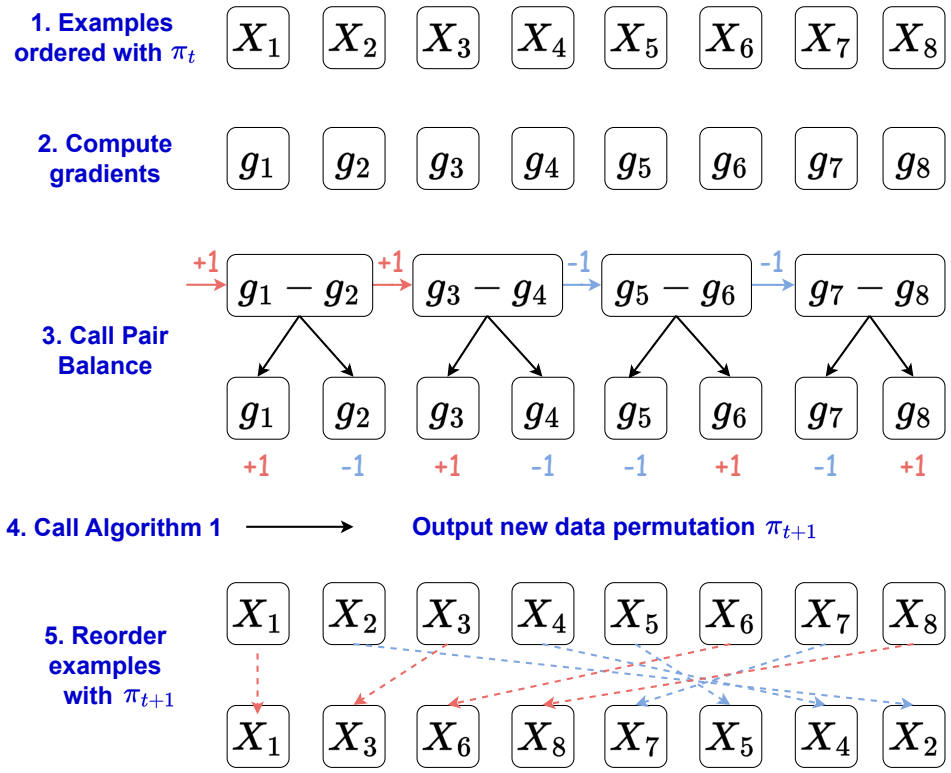


Figure 5: Schematic representation of online PairBalance for centralized GraB.

We also provide empirical results comparing GraB's Balance routine to the online PairBalance routine that we instead use in this work. We observe that both PairBalance and Balance would have similar convergence rates under centralized settings, and both outperform RR.

This experiment justifies the uses of PairBalance even in centralized learning settings. PairBalance theoretically tolerates higher learning rates and, as we will justify in Appendix D.1.2, is more memory-efficient than Balance. In short, PairBalance an excellent substitute for Balance when running GraB.

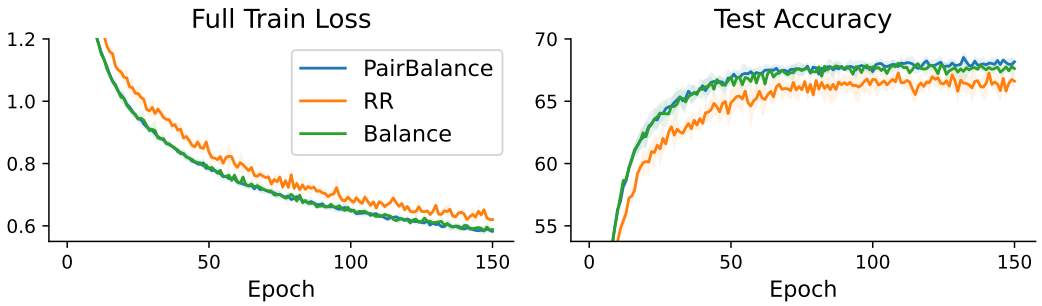


Figure 6: Convergence for centralized online PairBalance on LeNet on CIFAR-10. We use the identical set of hyperparameters (α = 1e-3, weight decay = 1e-2, momentum = 0.9, $B = 64$) as in the scaling experiments as in Figure 4.

C Proof Results

We present supporting results, which we use to prove the main results presented in Section 4. First, we show how to analyze the parallel herding bound in terms of a single step over the server-side PairBalance algorithm (Appendix C.1). We then include some additional observations/notation (Appendix C.2), which we use in the remaining intermediate results. We prove some intermediate results about how much the loss can change over the course of one epoch, assuming smoothness (Appendix C.3) and bounded gradient variance and heterogeneity (Appendix C.4). We combine these results to get one more intermediate result about the maximum the loss can change on average over many epochs (Appendix C.5), which we then use altogether to prove the two theorems that we present in the main paper (Appendix C.6).

C.1 Analyzing the parallel herding bound

In the main paper, we cover how CD-GraB runs on both the worker- and server-side. In this section, we dive deeper into the example-ordering part of CD-GraB, and demonstrate in theory how server-side online PairBalance reduces the parallel herding bound (8), as formulated in Section 3. We conclude this section by presenting Lemma 1, which shows server-side PairBalance is able to iteratively reduce the parallel herding bound.

To begin, we formalize our illustration over a group of vectors (since vector balancing, including PairBalance, does not inherently involve an optimization context until we use it in our online setting on gradients). Without loss of generality, we assume that the N examples are divided evenly among the m workers and that n is even. That is, we consider that we are given a set of vectors $\boldsymbol{z}_{i,j} \in \mathbb{R}^d$ for $i \in [m]$ and $j \in [n]$ evenly located on m workers (i.e., $n = \frac{N}{m}$), where $\boldsymbol{z}_{i,j}$ denotes the j-th vector located on the i-th worker. Now denote π_i as the original permutation of the vectors on worker i. Consider running Algorithm 7 on the server side over these N vectors.

Algorithm 7 Server-side PairBalance over a set of vectors (one step)

require: m workers, $n := \frac{N}{m}$ vectors per worker
input: initial permutations for all the workers $\{\pi_i\}_{i=1}^m$
initialize: new permutations for all the workers $\{\pi_i'\}_{i=1}^m$
initialize: running partial sum $\boldsymbol{h} = \mathbf{0}$
initialize: new indices front (left) pointer $\{l_i = 1\}_{i=1}^m$
initialize: new indices back (right) pointer $\{r_i = 1\}_{i=1}^m$
for example $j := 1 ... n$ **do**
  for worker $i := 1 ... m$ **do**
    if $j \bmod 2 = 0$ **then** ▷ If at an even index, i.e., can examine a full pair of examples
      $\boldsymbol{h}, s_{j-1}^i, s_j^i \leftarrow \mathsf{PairBalance}(\boldsymbol{h}, \boldsymbol{z}_{j-1}^i, \boldsymbol{z}_j^i)$
      if $s_{j-1}^i = +1$ **then**
        $\pi_i'(l_i) = j-1$; $l_i = l_i + 1$ ▷ Append first in pair to the front/left
        $\pi_i'(r_i) = j$; $r_i = r_i - 1$ ▷ Append second in pair to the right/back
      else
        $\pi_i'(l_i) = j$; $l_i = l_i + 1$ ▷ Append second in pair to the left/front
        $\pi_i'(r_i) = j-1$; $r_i = r_i - 1$ ▷ Append first in pair to the right/back
      end if
    end if
  end for
end for
output: new permutations for all m workers $\{\pi_i'\}_{i=1}^m$

Figure 7: One-step PairBalance algorithm on the server side to solve the parallel herding problem (8). This algorithm can be seen as a prototype for Algorithms 3 and 4, without the optimization context.

It follows, in the following Lemma 1, that we can get the parallel herding bound with the output permutations $\{\pi_i'\}_{i=1}^m$ from Algorithm 7:

Lemma 1. *Suppose that we have a set of vectors $\boldsymbol{z}_{i,j} \in \mathbb{R}^d$ for all $i,i' \in [m]$ and for all $j,j' \in [n]$ that satisfies*

$$\left\| \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{z}_{i,j} \right\|_\infty \leq c_1 \qquad \textit{and} \qquad \left\| \boldsymbol{z}_{i',j'} - \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{z}_{i,j} \right\|_\infty \leq c_2$$

for some constants $c_1 > 0$ and $c_2 > 0$. If we run Algorithm 7 over these vectors, then, for any $\delta > 0$, it holds with probability at least $1-\delta$ that

$$\max_{l \in [n]} \left\| \sum_{i=1}^{m} \sum_{j=1}^{l} \boldsymbol{z}_{i,\pi'_i(j)} \right\|_\infty \leq \frac{1}{2} \max_{l \in [n]} \left\| \sum_{i=1}^{m} \sum_{j=1}^{l} \boldsymbol{z}_{i,\pi_i(j)} \right\|_\infty + c_1 + \tilde{A} c_2,$$

where $\tilde{A}$ comes from Theorem 1.

Lemma 1 shows that PairBalance reduces the parallel herding objective (8) towards a constant (invariant to n) at each step. This implies that, if we repeatedly call PairBalance on a given permutation, it will return a permutation that guarantees the parallel herding bound to be $\tilde{O}(1)$.

Proof. We prove this lemma by defining the following auxiliary sequence of pair differences, as in Section 3.2

$$\boldsymbol{y}_{n \cdot (k-1)+i} = \boldsymbol{z}_{i,\pi_i(2k-1)} - \boldsymbol{z}_{i,\pi_i(2k)}, \, \forall k \in [n/2],$$

which we also can refer to as $\{\boldsymbol{y}_j\}_{j=1}^{mn/2}$.

We also leverage Theorem 1, which we reprint below for clarity of presentation:

Theorem 1 (Corollary 7, Dwivedi and Mackey [10]). *Consider any vectors $\{\boldsymbol{z}_j\}_{j=1}^N$ ($\boldsymbol{z}_j \in \mathbb{R}^d$) with $\|\boldsymbol{z}_j\|_2 \leq 1$ supplied as input to the* RandomizedBalance *subroutine in Algorithm 2. Then for any $\delta > 0$, with probability at least $1-\delta$,* RandomizedBalance *outputs a sequence of signs $\{s_j\}_{j=1}^N \in \{-1,1\}$ that satisfy $\max_{k \in [N]} \left\| \sum_{j=1}^{k} s_j \boldsymbol{z}_j \right\|_\infty \leq \tilde{A}$, where $\tilde{A} = \sqrt{2 \log\left(\frac{4d}{\delta}\right) \log\left(\frac{4N}{\delta}\right)} = \tilde{O}(1)$.*

Note that the reordering part of Algorithm 7 (line 8) gives a sequence of signs $\{s_j\}_{j=1}^{mn/2}$. Therefore, by Theorem 1, the sequence $\{\boldsymbol{y}_j\}_{j=1}^{mn/2}$ satisfies

$$\max_{P \in [mn/2]} \left\| \sum_{p=1}^{P} s_p \boldsymbol{y}_p \right\|_\infty \leq 2 \tilde{A} c_2, \tag{9}$$

since (based on what is given in Lemma 1)

$$\left\| \boldsymbol{y}_{n(k-1)+i} \right\|_\infty \leq \left\| \boldsymbol{z}_{i,\pi_{t,i}(2k-1)} - \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{z}_{i,j} \right\|_\infty + \left\| \boldsymbol{z}_{i,\pi_{t,i}(2k)} - \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{z}_{i,j} \right\|_\infty \leq 2 c_2.$$

Note that, if $s_{i,k}$ is the sign associated with $\boldsymbol{y}_{n(k-1)+i}$, then $\boldsymbol{z}_{i,\pi_{t,i}(2k-1)}$ and $\boldsymbol{z}_{i,\pi_{t,i}(2k)}$ will receive opposite signs $s_{i,k}$ and $-s_{i,k}$, respectively.

We denote $\boldsymbol{x}_{i,k}^+$ to be the example that receives sign $s_{i,k} = +1$ and $\boldsymbol{x}_{i,k}^-$ to be the example that receives sign $s_{i,k} = -1$.

That is, if $s_{i,k} = +1$, then $\boldsymbol{x}_{i,k}^+ = \boldsymbol{z}_{i,\pi_i(2k-1)}$, otherwise, if $s_{i,k} = -1$, then $\boldsymbol{x}_{i,k}^+ = \boldsymbol{z}_{i,\pi_i(2k)}$; and, $\boldsymbol{x}_{i,k}^-$ is the other term of the pair $\{\boldsymbol{z}_{i,\pi_i(2k-1)}, \boldsymbol{z}_{i,\pi_i(2k)}\}$.

Now, for $K \in [\frac{n}{2}]$, let

$$\kappa_{i,K} = \sum_{k=1}^{K} (\boldsymbol{z}_{i,\pi_i(2k-1)} + \boldsymbol{z}_{i,\pi_i(2k)}) \text{ and}$$
$$\upsilon_{i,K} = \sum_{k=1}^{K} (s_{i,k}\boldsymbol{z}_{i,\pi_i(2k-1)} - s_{i,k}\boldsymbol{z}_{i,\pi_i(2k)}).$$

Then

$$\sum_{k=1}^{K} \boldsymbol{x}_{i,k}^{+} = \frac{1}{2}(\kappa_{i,K} + \upsilon_{i,K}) \quad \text{and} \quad \sum_{k=1}^{K} \boldsymbol{x}_{i,k}^{-} = \frac{1}{2}(\kappa_{i,K} - \upsilon_{i,K}).$$

Now, observe that

$$\sum_{i=1}^{m} \kappa_{i,K} = \sum_{j=1}^{2K}\sum_{i=1}^{m} \boldsymbol{z}_{i,\pi_i(j)} \quad \text{and} \quad \sum_{i=1}^{m} \upsilon_{i,K} = \sum_{p=1}^{mK} s_p \boldsymbol{y}_p.$$

Therefore,

$$\begin{aligned}
\max_{K\in[n/2]} \left\| \sum_{k=1}^{K}\sum_{i=1}^{m} \boldsymbol{x}_{i,k}^{+} \right\|_\infty &\le \frac{1}{2}\left(\max_{K\in[n/2]} \left\| \sum_{i=1}^{m} \kappa_{K,i} \right\|_\infty + \max_{K\in[n/2]} \left\| \sum_{i=1}^{m} \upsilon_{K,i} \right\|_\infty \right) \\
&\le \frac{1}{2} \max_{K\in[n/2]} \left\| \sum_{j=1}^{2K}\sum_{i=1}^{m} \boldsymbol{z}_{i,j} \right\|_\infty + \tilde{A}c_2 \quad \text{By substituting above and (9)} \\
&\le \frac{1}{2} \max_{k\in[n]} \left\| \sum_{j=1}^{k}\sum_{i=1}^{m} \boldsymbol{z}_{i,j} \right\|_\infty + \tilde{A}c_2.
\end{aligned}$$

And similarly,

$$\begin{aligned}
\max_{K\in[n/2]} \left\| \sum_{k=1}^{K}\sum_{i=1}^{m} \boldsymbol{x}_{i,k}^{-} \right\| &\le \frac{1}{2}\left(\max_{K\in[n/2]} \left\| \sum_{i=1}^{m} \kappa_{K,i} \right\|_\infty + \max_{K\in[n/2]} \left\| \sum_{i=1}^{m} \upsilon_{K,i} \right\|_\infty \right) \\
&\le \frac{1}{2} \max_{k\in[n]} \left\| \sum_{j=1}^{k}\sum_{i=1}^{m} \boldsymbol{z}_{i,j} \right\|_\infty + \tilde{A}c_2.
\end{aligned}$$

Applying the new permutation $\pi_i'(j)$ on the vectors $\boldsymbol{z}_{i,\pi_i(j)}$, we get for each $i \in [m]$ the permuted sequence

$$\boldsymbol{x}_{i,1}^{+}, \ldots, \boldsymbol{x}_{i,n/2}^{+}, \boldsymbol{x}_{i,n/2}^{-}, \ldots, \boldsymbol{x}_{i,1}^{-}.$$

Thus, we need to bound the herding objective of the sequence

$$\sum_{i=1}^{m} \boldsymbol{x}_{i,1}^{+}, \ldots, \sum_{i=1}^{m} \boldsymbol{x}_{i,n/2}^{+}, \sum_{i=1}^{m} \boldsymbol{x}_{i,n/2}^{-}, \ldots, \sum_{i=1}^{m} \boldsymbol{x}_{i,1}^{-}.$$

If the partial sums above peak at $t_0 \le n/2$, then we can bound the parallel herding objective as

$$\left\| \sum_{k=1}^{t_0}\sum_{i=1}^{m} \boldsymbol{x}_{i,k}^{+} \right\|_\infty = \max_{K\in[n/2]} \left\| \sum_{k=1}^{K}\sum_{i=1}^{m} \boldsymbol{x}_{i,k}^{+} \right\|_\infty \le \frac{1}{2} \max_{k\in[n]} \left\| \sum_{j=1}^{k}\sum_{i=1}^{m} \boldsymbol{z}_{i,j} \right\|_\infty + \tilde{A}c_2;$$

otherwise, we can bound the parallel herding objective as

$$\left\|\sum_{j=1}^{n}\sum_{i=1}^{m}\boldsymbol{z}_{i,j}-\sum_{k=1}^{m-t_0}\sum_{i=1}^{m}\boldsymbol{x}_{i,k}^{-}\right\|_\infty \leq \left\|\sum_{j=1}^{n}\sum_{i=1}^{m}\boldsymbol{z}_{i,j}\right\|_\infty + \left\|\sum_{k=1}^{m-t_0}\sum_{i=1}^{m}\boldsymbol{x}_{i,k}^{-}\right\|_\infty$$
$$\leq c_1 + \frac{1}{2}\max_{t\in[n]}\left\|\sum_{j=1}^{t}\sum_{i=1}^{m}\boldsymbol{z}_{i,j}\right\|_\infty + \tilde{A}c_2,$$

since in Algorithm 1 the list of vectors with negative signs is reversed before concatenated.

The claim follows. □

C.2 Notation and observations

We begin with three notes that we will use throughout the intermediate results we present in this section. We will use the lemmas presented here to prove our main results: Theorems 2 and 3 in Appendix C.6.

1. **A single t-th update.** First, recall that one t-th step of the parameter update can be written as
$$\boldsymbol{w}_t^{j+1} = \boldsymbol{w}_t^j - \frac{\alpha}{m}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_t^j;\pi_{t,i}(j)), \quad \forall j\in[n]$$
We will use the convention $\boldsymbol{w}_{t+1} \triangleq \boldsymbol{w}_{t+1}^1 \triangleq \boldsymbol{w}_t^{n+1}$.
2. **The maximum amount a parameter can change over an epoch.** The key quantity in our proof is Δ_t, which is the maximum amount that a parameter in $\boldsymbol{w}$ can change in epoch t. That is,
$$\Delta_t \triangleq \max_{k\in[n]}\left\|\boldsymbol{w}_t^{k+1}-\boldsymbol{w}_t\right\|_\infty$$
$$= \frac{\alpha}{m}\max_{k\in[n]}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_t^j;\pi_{t,i}(j))\right\|_\infty. \tag{10}$$
Following this definition of Δ_t, we note that the maximum amount that a parameter in $\boldsymbol{w}$ can change over two different epochs is $2\Delta_t$. That is, we observe
$$\left\|\boldsymbol{w}_t^j-\boldsymbol{w}_t^k\right\|_\infty \leq \left\|\boldsymbol{w}_t^j-\boldsymbol{w}_t\right\|_\infty + \left\|\boldsymbol{w}_t^k-\boldsymbol{w}_t\right\|_\infty \leq 2\Delta_t$$
$$\left\|\boldsymbol{w}_{t+1}^j-\boldsymbol{w}_t^k\right\|_\infty \leq \left\|\boldsymbol{w}_{t+1}^j-\boldsymbol{w}_{t+1}\right\|_\infty + \left\|\boldsymbol{w}_{t+1}-\boldsymbol{w}_t\right\|_\infty + \left\|\boldsymbol{w}_t^k-\boldsymbol{w}_t\right\|_\infty \leq \Delta_{t+1}+2\Delta_t, \quad \forall j,k\in[n].$$
We make repeated use of this relation in the results that follow, which we typically will use in combination with the Lipschitz assumption to bound gradients of the same loss function but with different parameters.
3. **Bounding loss at epoch t.** We will denote $F_t = f(\boldsymbol{w}_t) - f(\boldsymbol{w}^*)$ where $\boldsymbol{w}^*$ is the minimizer of f which we assume to be bounded from below.

C.3 Assuming $L_{2,\infty}$-smoothness: results on the amount the loss can change over one epoch)

We will next prove an intermediate result regarding that bounds the loss f at epoch $t+1$ in relation to the loss at the prior epoch t (Lemma 2). That is, we prove results about how much the loss with respect to the parameters can change over the course of one epoch.

Lemma 2. *If the loss f is $L_{2,\infty}$-smooth and the learning rate $\alpha \leq \frac{1}{nL_{2,\infty}}$, then*
$$f(\boldsymbol{w}_{t+1}) \leq f(\boldsymbol{w}_t) + \frac{\alpha n L_{2,\infty}^2}{2}\Delta_t^2 - \frac{\alpha n}{2}\|\nabla f(\boldsymbol{w}_t)\|_2^2. \tag{11}$$

Proof. We begin with the definition of $L_{2,\infty}$-smoothness, with respect to loss f:
$$f(\boldsymbol{w}_{t+1}) \leq f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w}_{t+1}-\boldsymbol{w}_t) + \frac{L_{2,\infty}}{2}\|\boldsymbol{w}_{t+1}-\boldsymbol{w}_t\|_2^2$$

Also observe that

$$-\nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t-\boldsymbol{w}_{t+1}) = -\frac{\alpha n}{2} 2\nabla f(\boldsymbol{w}_t)^\top\left(\frac{\boldsymbol{w}_t-\boldsymbol{w}_{t+1}}{\alpha n}\right)$$
$$= \frac{\alpha n}{2}\left(\left\|\nabla f(\boldsymbol{w}_t)-\frac{(\boldsymbol{w}_t-\boldsymbol{w}_{t+1})}{\alpha n}\right\|_2^2 - \|\nabla f(\boldsymbol{w}_t)\|_2^2 - \left\|\frac{(\boldsymbol{w}_t-\boldsymbol{w}_{t+1})}{\alpha n}\right\|_2^2\right). \tag{12}$$

Combining the above — i.e., the definition of $L_{2,\infty}$-smoothness with (12) — we get

$$f(\boldsymbol{w}_{t+1}) \le f(\boldsymbol{w}_t) + \frac{\alpha n}{2}\left\|\nabla f(\boldsymbol{w}_t)-\frac{(\boldsymbol{w}_t-\boldsymbol{w}_{t+1})}{\alpha n}\right\|_2^2 - \frac{\alpha n}{2}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \qquad + \frac{\alpha n L_{2,\infty}-1}{2\alpha n}\|\boldsymbol{w}_t-\boldsymbol{w}_{t+1}\|_2^2.$$

The last term on the right-hand side is ≤ 0 by the assumption that the learning rate $\alpha \le \frac{1}{nL_{2,\infty}}$. Therefore,

$$f(\boldsymbol{w}_{t+1}) \le f(\boldsymbol{w}_t) + \frac{\alpha n}{2}\left\|\nabla f(\boldsymbol{w}_t)-\frac{(\boldsymbol{w}_t-\boldsymbol{w}_{t+1})}{\alpha n}\right\|_2^2 - \frac{\alpha n}{2}\|\nabla f(\boldsymbol{w}_t)\|_2^2. \tag{13}$$

We next bound the second term on the right-hand side by Δ_t (10):

$$\left\|\nabla f(\boldsymbol{w}_t)-\frac{(\boldsymbol{w}_t-\boldsymbol{w}_{t+1})}{\alpha n}\right\|_2^2 = \left\|\frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\nabla f^i(\boldsymbol{w}_t,\pi_t(j)) - \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\nabla f^i(\boldsymbol{w}_t^j;\pi_t(j))\right\|_2^2$$
$$\le \frac{1}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\left\|\nabla f^i(\boldsymbol{w}_t,\pi_t(j)) - \nabla f^i(\boldsymbol{w}_t^j;\pi_t(j))\right\|_2^2$$
$$\le \frac{L_{2,\infty}^2}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\left\|\boldsymbol{w}_t^j-\boldsymbol{w}_t\right\|_\infty^2,$$

where we have used $L_{2,\infty}$-smoothness (Assumption 3) in the last inequality. Substituting Δ_t, we get

$$\frac{L_{2,\infty}^2}{mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\left\|\boldsymbol{w}_t^j-\boldsymbol{w}_t\right\|_\infty^2 \quad \le \quad L_{2,\infty}^2\Delta_t^2.$$

Plugging the above into (13), we get

$$f(\boldsymbol{w}_{t+1}) \le f(\boldsymbol{w}_t) + \frac{\alpha n L_{2,\infty}^2}{2}\Delta_t^2 - \frac{\alpha n}{2}\|\nabla f(\boldsymbol{w}_t)\|_2^2,$$

yielding the claim. □

We next build slightly on Lemma 2 to make two additional observations. First:

Lemma 3. *If the loss f is $L_{2,\infty}$-smooth and the learning rate $\alpha \le \frac{1}{nL_{2,\infty}}$, then*

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \le \frac{2F_1}{\alpha n T} + \frac{L_{2,\infty}^2}{T}\sum_{t=1}^{T}\Delta_t^2,$$

where F_1 comes from Theorem 2.

Proof. Using Lemma 2 and Jensen's inequality, we average (11) over $t \in [T]$ and match terms, yielding

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \le \frac{2(f(\boldsymbol{w}_1)-f(\boldsymbol{w}_{T+1}))}{\alpha n T} + \frac{L_{2,\infty}^2}{T}\sum_{t=1}^{T}\Delta_t^2.$$

Substituting F_1, we get

$$\le \frac{2F_1}{\alpha n T} + \frac{L_{2,\infty}^2}{T}\sum_{t=1}^{T}\Delta_t^2,$$

yielding the claim. □

We next build on Lemma 2 by further assuming the P.L. assumption holds.

Lemma 4. *If the loss f is $L_{2,\infty}$-smooth, the learning rate $\alpha \leq \frac{1}{nL_{2,\infty}}$, and the P.L. assumption (Assumption 4) holds, then, for $\rho = 1 - \frac{\alpha n \mu}{2}$*

$$F_{T+1} \leq \rho^T F_1 + \frac{\alpha n L_{2,\infty}^2}{2} \sum_{t=1}^{T} \rho^{T-t} \left(\Delta_t^2 - \frac{1}{2L_{2,\infty}^2} \|\nabla f(\boldsymbol{w}_t)\|_2^2 \right).$$

Proof. From Lemma 2, we got (11), i.e.,

$$f(\boldsymbol{w}_{t+1}) \leq f(\boldsymbol{w}_t) + \frac{\alpha n L_{2,\infty}^2}{2} \Delta_t^2 - \frac{\alpha n}{2} \|\nabla f(\boldsymbol{w}_t)\|_2^2,$$

Applying the P.L. assumption (Assumption 4) to (11), we get

$$\begin{aligned} f(\boldsymbol{w}_{t+1}) &\leq f(\boldsymbol{w}_t) + \frac{\alpha n L_{2,\infty}^2}{2} \Delta_t^2 - \frac{\alpha n}{4} \|\nabla f(\boldsymbol{w}_t)\|_2^2 - \frac{\alpha n}{4} \|\nabla f(\boldsymbol{w}_t)\|_2^2 \\ &\leq f(\boldsymbol{w}_t) + \frac{\alpha n L_{2,\infty}^2}{2} \Delta_t^2 - \frac{\alpha n \mu}{2} (f(\boldsymbol{w}_t) - f(\boldsymbol{w}^*)) - \frac{\alpha n}{4} \|\nabla f(\boldsymbol{w}_t)\|_2^2. \end{aligned}$$

Subtracting f^* from both sides, we get

$$f(\boldsymbol{w}_{t+1}) - f^* \leq \left(1 - \frac{\alpha n \mu}{2}\right)(f(\boldsymbol{w}_t) - f^*) + \frac{\alpha n}{2} \left(L_{2,\infty}^2 \Delta_t^2 - \frac{1}{2} \|\nabla f(\boldsymbol{w}_t)\|_2^2 \right).$$

For $\rho = 1 - \frac{\alpha n \mu}{2}$, we then apply the above inequality recursively for $t \in [T]$, yielding the claim:

$$F_{T+1} \leq \rho^T F_1 + \frac{\alpha n L_{2,\infty}^2}{2} \sum_{t=1}^{T} \rho^{T-t} \left(\Delta_t^2 - \frac{1}{2L_{2,\infty}^2} \|\nabla f(\boldsymbol{w}_t)\|_2^2 \right).$$

□

C.4 Assuming bounded gradient variance and heterogeneity: results applying Algorithm 7

We next prove a result that builds on Lemma 1 and our one-step version of the server-side PairBalance algorithm (Algorithm 7).

We begin by introducing some additional notation. Namely, we will call π^{-1} the operation that, given an example, yields the index in the permutation for that example. For instance, $\pi_{t+1,i}(j)$ returns the example at the j-th index for the i-th worker's $t+1$ permutation. Let us denote that example τ. Then, $\pi_{t,i}^{-1} \pi_{t+1,i}(j)$ is equivalent to applying $\pi_{t,i}^{-1}$ to τ: it takes the example τ and returns τ's associated index in the i-th worker's epoch t's permutation (in this case, the prior epoch's permutation).

We will make use of this notation in the following Lemma.

Lemma 5. *Assume bounded gradient variance (Assumption 1), bounded gradient heterogeneity (Assumption 2), and $L_{2,\infty}$-smoothness (Assumption 3). For $t \in [T]$ and $\delta > 0$, if we apply Algorithm 7 to the gradients $\nabla f^i(\boldsymbol{w}_t^j; \pi_t^i(j))$ at epoch t to produce the next permutation $\pi_{t+1,i}$ for epoch $t+1$, then, with probability at least $1-\delta$,*

$$\Delta_{t+1} \quad \leq \quad \frac{1}{2}\Delta_t + \alpha L_{2,\infty} \left(4n + \frac{2\tilde{A}}{m} \right) \Delta_t + \alpha n L_{2,\infty} \Delta_{t+1} + \frac{\alpha(\varsigma + \sigma)\tilde{A}}{m} + \alpha n \|\nabla f(\boldsymbol{w}_{t+1})\|_2,$$

where $\tilde{A}$ comes from Theorem 1.

Proof. We start with the triangle inequality:

$$\begin{aligned} \left\| \sum_{j=1}^{k} \sum_{i=1}^{m} \nabla f^i(\boldsymbol{w}_{t+1}^j; \pi_{t+1,i}(j)) \right\|_\infty \leq & \left\| \sum_{j=1}^{k} \sum_{i=1}^{m} \nabla f^i(\boldsymbol{w}_t^{\pi_{t,i}^{-1} \pi_{t+1,i}(j)}; \pi_{t+1,i}(j)) \right\|_\infty + \\ & \left\| \sum_{j=1}^{k} \sum_{i=1}^{m} \left(\nabla f^i(\boldsymbol{w}_{t+1,i}^j, \pi_{t+1,i}(j)) - \nabla f^i(\boldsymbol{w}_{t,i}^{\pi_{t,i}^{-1} \pi_{t+1,i}(j)}; \pi_{t+1,i}(j)) \right) \right\|_\infty \end{aligned} \tag{14}$$

We use Lemma 1 to bound the first term on the right-hand side of (14) from Lemma 2.

That is, let

$$\boldsymbol{z}_{i,j} = \nabla f^i(\boldsymbol{w}_t^{\pi_{t,i}^{-1}(j)}; j),$$

so that

$$\boldsymbol{z}_{i,\pi_{t+1,i}(j)} = \nabla f^i(\boldsymbol{w}_t^{\pi_{t,i}^{-1}\pi_{t+1,i}(j)}; \pi_{t+1,i}(j)).$$

The upper bounds for $\left\|\boldsymbol{z}_{i,j} - \frac{1}{mn}\sum_{r,s}\boldsymbol{z}_{r,s}\right\|_\infty$ and $\left\|\sum_{i,j}\boldsymbol{z}_{i,j}\right\|_\infty$ are:

$$\left\|\nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j)) - \frac{1}{mn}\sum_{r=1}^{m}\sum_{s=1}^{n}\nabla f^s(\boldsymbol{w}_t^r; \pi_{t,s}(r))\right\|_\infty,$$

which are

$$\begin{aligned}
&\leq \left\|\nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j)) - \frac{1}{mn}\sum_{r=1}^{m}\sum_{s=1}^{n}\nabla f^s(\boldsymbol{w}_t^j; \pi_{t,s}(r))\right\|_\infty + \\
&\quad \left\|\frac{1}{mn}\sum_{r=1}^{m}\sum_{s=1}^{n}\nabla f^s(\boldsymbol{w}_t^j; \pi_{t,s}(r)) - \frac{1}{mn}\sum_{r=1}^{m}\sum_{s=1}^{n}\nabla f^s(\boldsymbol{w}_t^r; \pi_{t,s}(r))\right\|_\infty.
\end{aligned}$$

We can rewrite the above to be

$$\begin{aligned}
&\leq \left\|\nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j)) - \nabla f(\boldsymbol{w}_t^j)\right\|_\infty + \frac{L_{2,\infty}}{mn}\sum_{r=1}^{m} m \sum_{s=1}^{n}\left\|\boldsymbol{w}_t^j - \boldsymbol{w}_t^r\right\|_\infty \\
&\leq \varsigma + \sigma + 2L_{2,\infty}\Delta_t,
\end{aligned}$$

by Assumptions 1, 2, and 3, and by the definition of Δ_t (10).

Now, observe that

$$\begin{aligned}
\left\|\sum_{i=1}^{m}\sum_{j=1}^{n}\nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j))\right\|_\infty \leq &\left\|\sum_{i=1}^{m}\sum_{j=1}^{n}\nabla f^i(\boldsymbol{w}_t^j; \pi_{t,i}(j)) - \sum_{i=1}^{m}\sum_{j=1}^{n}\nabla f^i(\boldsymbol{w}_{t+1}; \pi_{t,i}(j))\right\|_\infty + \\
&\left\|\sum_{i=1}^{m}\sum_{j=1}^{n}\nabla f^i(\boldsymbol{w}_{t+1}; \pi_{t,i}(j))\right\|_\infty.
\end{aligned}$$

By using the above, we can rewrite the right-hand side to be

$$\begin{aligned}
&\leq \sum_{i=1}^{m}\sum_{j=1}^{n} L_{2,\infty}\left\|\boldsymbol{w}_t^j - \boldsymbol{w}_{t+1}\right\|_\infty + mn\|\nabla f(\boldsymbol{w}_{t+1})\|_\infty \\
&\leq 2mnL_{2,\infty}\Delta_t + mn\|\nabla f(\boldsymbol{w}_{t+1})\|_2.
\end{aligned}$$

Therefore, by Lemma 1,

$$\begin{aligned}
\max_{k\in[n]}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_t^{\pi_{t,i}^{-1}\pi_{t+1,i}(j)}, \pi_{t+1,i}(j))\right\|_\infty \leq &\max_{k\in[n]}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_t^j, \pi_{t,i}(j))\right\|_\infty \\
&+ 2mnL_{2,\infty}\Delta_t + \|\nabla f(\boldsymbol{w}_{t+1})\|_2 + (\varsigma + \sigma + 2L_{2,\infty}\Delta_t)\tilde{A}.
\end{aligned}$$

The second term of the triangle inequality (14) can be bounded as

$$\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\left(\nabla f^i(\boldsymbol{w}_{t+1,i}^j; \pi_{t+1,i}(j)) - \nabla f^i(\boldsymbol{w}_{t,i}^{\pi_{t,i}^{-1}\pi_{t+1,i}(j)}; \pi_{t+1,i}(j))\right)\right\|_\infty,$$

which is

$$\leq \sum_{j=1}^{k}\sum_{i=1}^{m}\left\|\boldsymbol{w}_{t+1,i}^{j}-\boldsymbol{w}_{t,i}^{\pi_{t,i}^{-1}\pi_{t+1,i}(j)}\right\|_{\infty}$$
$$\leq mnL_{2,\infty}(\Delta_{t+1}+2\Delta_t).$$

Substituting these bounds into the right-hand side of the triangle inequality (14), taking the max of both sides, and grouping terms, we get

$$\max_{k\in[n]}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_{t+1,i}^{j},\pi_{t+1,i}(j))\right\|_{\infty} \leq \frac{1}{2}\max_{k\in[n]}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_{t}^{j},\pi_{t,i}(j))\right\|_{\infty} + L_{2,\infty}(4mn+2\tilde{A})\Delta_t + mnL_{2,\infty}\Delta_{t+1} + (\varsigma+\sigma)\tilde{A} + mn\|\nabla f(\boldsymbol{w}_{t+1})\|_2.$$

Multiplying both sides by $\frac{\alpha}{m}$ and using the definition of Δ_t (10), we get the claim. □

C.5 Combining the prior intermediate results: proofs over multiple steps

Lemma 6. *If the learning rate* $\alpha \leq \frac{1}{16L_{2,\infty}(2n+\tilde{A}/m)}$*, then*

$$\frac{1}{T}\sum_{t=1}^{T}\Delta_t^2 \leq \frac{21\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2}+\frac{9\alpha^2n^2\sigma^2}{T}+21\alpha^2n^2\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2.$$

Proof. First, we bound Δ_1^2.

We start with a series of triangle inequalities:

$$\frac{\alpha}{m}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_1^j,\pi_{1,i}(j))\right\|_{\infty} \leq \frac{\alpha}{m}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_1^j,\pi_{1,i}(j))-\sum_{j=1}^{k}\sum_{i=1}^{m}\nabla f^i(\boldsymbol{w}_1,\pi_{1,i}(j))\right\|_{\infty} + \frac{\alpha}{m}\left\|\sum_{j=1}^{k}\sum_{i=1}^{m}\big(\nabla f^i(\boldsymbol{w}_1,\pi_{1,i}(j))-\nabla f^i(\boldsymbol{w}_1)\big)\right\|_{\infty}+\alpha k\|\nabla f(\boldsymbol{w}_1)\|_{\infty}$$
$$\leq \frac{\alpha}{m}\sum_{j=1}^{k}\sum_{i=1}^{m}L_{2,\infty}\left\|\boldsymbol{w}_1^j-\boldsymbol{w}_1\right\|_{\infty}+\alpha k\sigma+\alpha k\|\nabla f(\boldsymbol{w}_1)\|_2.$$

We next take the max of both sides with respect to $k\in[n]$:

$$\begin{aligned}\Delta_1 &\leq \alpha nL_{2,\infty}\Delta_1+\alpha n\sigma+\alpha n\|\nabla f(\boldsymbol{w}_1)\|_2\\ &\leq (1/32)\Delta_1+\alpha n\sigma+\alpha n\|\nabla f(\boldsymbol{w}_1)\|_2 \qquad (\text{since } \alpha\leq\frac{1}{32nL_{2,\infty}})\\ &\leq (32/31)\alpha n\sigma+(32/31)\alpha n\|\nabla f(\boldsymbol{w}_1)\|_2,\end{aligned}$$

Squaring both sides:

$$\Delta_1^2 \leq 3\alpha^2n^2\sigma^2+3\alpha^2n^2\|\nabla f(\boldsymbol{w}_1)\|_2^2. \tag{15}$$

Now, we use Lemma 5 to get the relationship between Δ_{t+1} and Δ_t for $t\in[T]$.

Recall that

$$\Delta_{t+1}\leq\frac{1}{2}\Delta_t+\alpha L_{2,\infty}\left(4n+\frac{2\tilde{A}}{m}\right)\Delta_t+\alpha nL_{2,\infty}\Delta_{t+1}+\frac{\alpha(\varsigma+\sigma)\tilde{A}}{m}+\alpha n\|\nabla f(\boldsymbol{w}_{t+1})\|_2$$

Because $\alpha \leq \frac{1}{16L_{2,\infty}(2n+\tilde{A}/m)}$, we can rewrite the above as

$$\Delta_{t+1} \leq \frac{1}{2}\Delta_t + (1/8)\Delta_t + (1/32)\Delta_{t+1} + \frac{\alpha(\varsigma+\sigma)\tilde{A}}{m} + \alpha n \|\nabla f(\boldsymbol{w}_{t+1})\|_2.$$

Squaring both sides:

$$\begin{aligned}(31/32)^2\Delta_{t+1}^2 &\leq \frac{1}{2}\Delta_t^2 + 2\left((1/8)\Delta_t + \frac{\alpha(\varsigma+\sigma)\tilde{A}}{m} + \alpha n\|\nabla f(\boldsymbol{w}_{t+1})\|_2\right)^2 \\ &\leq \frac{1}{2}\Delta_t^2 + (6/8^2)\Delta_t^2 + \frac{6\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + 6\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_{t+1})\|_2^2,\end{aligned}$$

so that

$$\begin{aligned}\Delta_{t+1}^2 &\leq (32/31)^2(1/2+6/8^2)\Delta_t^2 + \frac{(32/31)^2 6\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + (32/31)^2 6\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_{t+1})\|_2^2 \\ &\leq (2/3)\Delta_t^2 + \frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + 7\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_{t+1})\|_2^2.\end{aligned} \tag{16}$$

We next sum (16) over $t \in [T-1]$ and add (15):

$$\begin{aligned}\Delta_1^2 + \sum_{t=2}^{T}\Delta_t^2 &\leq (2/3)\sum_{t=2}^{T}\Delta_{t-1}^2 + \frac{(T-1)7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + 3\alpha^2 n^2\sigma^2 + 7\alpha^2 n^2 \sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \\ \frac{1}{T}\sum_{t=1}^{T}\Delta_t^2 &\leq (2/3)\frac{1}{T}\sum_{t=1}^{T}\Delta_t^2 + \frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + \frac{3\alpha^2 n^2\sigma^2}{T} + 7\alpha^2 n^2 \frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \\ &\leq \frac{21\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + \frac{9\alpha^2 n^2\sigma^2}{T} + 21\alpha^2 n^2 \frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2,\end{aligned}$$

yielding the claim. □

We next build on Lemma 6.

Lemma 7. *If $\alpha \leq \frac{2}{9n\mu}$, then, for $\rho = 1 - \frac{\alpha n\mu}{2}$,*

$$\sum_{t=1}^{T}\rho^{T-t}\Delta_t^2 \leq 12\rho^{T-1}\alpha^2 n^2\sigma^2 + \frac{28\rho\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{(1-\rho)m^2} + \frac{1}{2L_{2,\infty}^2}\sum_{t=1}^{T}\rho^{T-t}\|\nabla f(\boldsymbol{w}_t)\|_2^2.$$

Proof. Recall (15) from Lemma 6:

$$\Delta_1^2 \leq 3\alpha^2 n^2\sigma^2 + 3\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_1)\|_2^2.$$

We multiply each term Δ_t with ρ^{T-t} for $t \in [T]$ and get

$$\rho^{T-1}\Delta_1^2 \leq \rho^{T-1}3\alpha^2 n^2\sigma^2 + \rho^{T-1}3\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_1)\|_2^2. \tag{17}$$

Similarly, recall (16) from Lemma 6,

$$\Delta_{t+1}^2 \leq (2/3)\Delta_t^2 + \frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + 7\alpha^2 n^2 \|\nabla f(\boldsymbol{w}_{t+1})\|_2^2,$$

for which we also multiply each term Δ_t with ρ^{T-t} for $t \in [T]$, and get

$$\rho^{T-t}\Delta_t^2 \leq (2/3)\rho^{T-t}\Delta_{t-1}^2 + \rho^{T-t}\frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + \rho^{T-t}7\alpha^2 n^2\|\nabla f(\boldsymbol{w}_t)\|_2^2$$
$$\leq (3/4)\rho^{T-(t-1)}\Delta_{t-1}^2 + \rho^{T-t}\frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} + \rho^{T-t}7\alpha^2 n^2\|\nabla f(\boldsymbol{w}_t)\|_2^2, \quad \forall t\in\{2,...,T\}, \tag{18}$$

where we have used $\alpha \leq \frac{2}{9n\mu}$ so that $\rho = 1-\frac{\alpha n\mu}{2} \geq (2/3)(4/3)$.

Next, we sum the bounds in (17) and (18) for $\rho^{T-t}\Delta_t$ for all $t\in[T]$, and we get

$$\rho^{T-1}\Delta_1^2 + \sum_{t=2}^{T}\rho^{T-t}\Delta_t^2 \leq \frac{3}{4}\sum_{t=2}^{T}\rho^{T-(t-1)}\Delta_{t-1}^2 + \rho^{T-1}3\alpha^2 n^2\sigma^2 + \sum_{t=1}^{T}\rho^{T-t}\frac{7\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{m^2} +$$
$$7\alpha^2 n^2\sum_{t=1}^{T}\rho^{T-t}\|\nabla f(\boldsymbol{w}_t)\|_2^2.$$

We can rewrite the right-hand side as

$$\leq \frac{3}{4}\sum_{t=1}^{T}\rho^{T-t}\Delta_t^2 + \rho^{T-1}3\alpha^2 n^2\sigma^2 + \frac{7\rho\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{(1-\rho)m^2} + 7\alpha^2 n^2\sum_{t=1}^{T}\rho^{T-t}\|\nabla f(\boldsymbol{w}_t)\|_2^2$$
$$\leq 12\rho^{T-1}\alpha^2 n^2\sigma^2 + \frac{28\rho\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{(1-\rho)m^2} + 28\alpha^2 n^2\sum_{t=1}^{T}\rho^{T-t}\|\nabla f(\boldsymbol{w}_t)\|_2^2.$$

Lastly, we use $\alpha \leq \frac{1}{\sqrt{56}nL_{2,\infty}}$ to get:

$$\sum_{t=1}^{T}\rho^{T-t}\Delta_t^2 \leq 12\rho^{T-1}\alpha^2 n^2\sigma^2 + \frac{28\rho\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{(1-\rho)m^2} + \frac{1}{2L_{2,\infty}^2}\sum_{t=1}^{T}\rho^{T-t}\|\nabla f(\boldsymbol{w}_t)\|^2.$$

□

C.6 Proof of Theorems 2 and 3

Using the Lemmas above, we next prove our main results, presented in Section 4.

Proof of Theorem 2. The given learning rate α satisfies the constraints of Lemma 3 and Lemma 6. Therefore,

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \leq \frac{2F_1}{\alpha nT} + L_{2,\infty}^2\left(\frac{21(\alpha(\varsigma+\sigma)\tilde{A})^2}{m^2} + \frac{9(\alpha n\sigma)^2}{T} + 21\alpha^2 n^2 \frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2\right)$$
$$\leq \frac{4F_1}{\alpha nT} + \frac{42L_{2,\infty}^2(\alpha(\varsigma+\sigma)\tilde{A})^2}{m^2} + \frac{18L_{2,\infty}^2(\alpha n\sigma)^2}{T},$$

due to $\alpha \leq \frac{1}{\sqrt{42}nL_{2,\infty}}$.

We next derive the convergence rate. Let $\Gamma = \frac{42(L_{2,\infty}(\varsigma+\sigma)\tilde{A})^2}{m^2} + \frac{18L_{2,\infty}^2 n^2\sigma^2}{T}$. Then,

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \leq \frac{4F_1}{\alpha nT} + \Gamma\alpha^2.$$

We then set $\alpha \leq \left(\frac{4F_1}{n\Gamma T}\right)^{1/3}$. So we will have $\alpha = \min\left\{\frac{1}{16L_{2,\infty}(2n+\tilde{A}/m)}, \left(\frac{4F_1}{n\Gamma T}\right)^{1/3}\right\}$ or

$$\frac{1}{\alpha} = \max\left\{16L_{2,\infty}(2n+\tilde{A}/m), \left(\frac{4F_1}{n\Gamma T}\right)^{-1/3}\right\}.$$

Substitute α:

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(\boldsymbol{w}_t)\|_2^2 \leq \frac{4F_1}{nT}\left\{16L_{2,\infty}(2n+\tilde{A}/m) + \left(\frac{4F_1}{n\Gamma T}\right)^{-1/3}\right\} + \Gamma\left(\frac{4F_1}{n\Gamma T}\right)^{2/3}$$
$$\leq \left(\frac{4F_1}{nT}\right)^{2/3}\Gamma^{1/3} + \frac{64F_1L_{2,\infty}(2+\tilde{A}/(mn))}{T}$$
$$\leq \left(\frac{4F_1}{nT}\right)^{2/3}\left(\frac{(\sqrt{42}L_{2,\infty}(\varsigma+\sigma)\tilde{A})^{2/3}}{m^{2/3}} + \frac{(\sqrt{18}L_{2,\infty}n\sigma)^{2/3}}{T^{1/3}}\right)$$
$$+ \frac{64F_1L_{2,\infty}(2+\tilde{A}/(mn))}{T}$$
$$\leq \frac{(4\sqrt{42}F_1L_{2,\infty}(\varsigma+\sigma)\tilde{A})^{2/3}}{(mnT)^{2/3}} + \frac{(72F_1L_{2,\infty}\sigma)^{2/3}}{T}$$
$$+ \frac{64F_1L_{2,\infty}(2+\tilde{A}/(mn))}{T},$$

Since $(4\sqrt{42})^{2/3} < 9$, the above is

$$\leq \frac{9(F_1L_{2,\infty}(\varsigma+\sigma)\tilde{A})^{2/3}}{(mnT)^{2/3}} + \frac{(72F_1L_{2,\infty}\sigma)^{2/3} + 64F_1L_{2,\infty}(2+\tilde{A}/(mn))}{T},$$

in which the leading term (slowest in terms of T) is $\tilde{O}((mnT)^{-2/3})$, proving the claim. □

Proof of Theorem 3. With the P.L. assumption (Assumption 4), we use Lemma 4 and Lemma 7. We show that their constraints are satisfied later) to get

$$\begin{aligned}
F_{T+1} &\leq \rho^T F_1 + \frac{\alpha n L_{2,\infty}^2}{2} \sum_{t=1}^{T} \rho^{T-t} \left(\Delta_t^2 - \frac{1}{2L_{2,\infty}^2} \|\nabla f(\boldsymbol{w}_t)\|_2^2 \right) \\
&\leq \rho^T F_1 + \frac{\alpha n L_{2,\infty}^2}{2} \left(12\rho^{T-1}\alpha^2 n^2 \sigma^2 + \frac{28\rho\alpha^2(\varsigma+\sigma)^2\tilde{A}^2}{(1-\rho)m^2} \right) \\
&\leq \rho^T F_1 + \rho^{T-1} 6\alpha^3 n^3 L_{2,\infty}^2 \sigma^2 + \frac{28\rho\alpha^3 n L_{2\infty}^2(\varsigma+\sigma)^2\tilde{A}^2}{\alpha n\mu m^2} \\
&\leq \rho^T F_1 + \rho^T 7\alpha^3 n^3 L_{2,\infty}^2 \sigma^2 + \frac{28\rho\alpha^3 n L_{2\infty}^2(\varsigma+\sigma)^2\tilde{A}^2}{\alpha n\mu m^2} \\
&\leq \rho^T (F_1 + \sigma^2/L_{2,\infty}) + \frac{28\alpha^2 L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2}{\mu m^2} \\
&\leq (F_1 + \sigma^2/L_{2,\infty}) \exp(-T\alpha n\mu/2) + \frac{28\alpha^2 L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2}{\mu m^2},
\end{aligned}$$

where we have further constrained $\alpha \leq \frac{2}{9n\mu}$ so that $\rho \leq 9/8$ in the forth inequality and $\alpha \leq \frac{1}{7^{1/3} n L_{2,\infty}}$ in the fifth inequality. By setting the derivative w.r.t α of the RHS to 0, the minimizer α under the constraint that $0 < \alpha \leq \min\left\{ \frac{2}{9n\mu}, \frac{1}{16 L_{2,\infty}(2n+\tilde{A}/m)} \right\}$ (required by the lemmas) is:

$$\alpha = \frac{2}{Tn\mu} W_0(T^2 m^2 n^2 C_3),$$

as long as

$$\begin{aligned}
T &\geq 1 + \frac{2}{n\mu} \max\{(9/2)n\mu, 16 L_{2,\infty}(2n+\tilde{A}/m) W_0(T^2 m^2 n^2 C_3)\} \\
&= 10 + \frac{1}{\mu} 32 L_{2,\infty}(2+\tilde{A}/(mn)) W_0(T^2 m^2 n^2 C_3),
\end{aligned}$$

where $C_3 = \frac{(F_1+\sigma^2/L_{2,\infty})\mu^2}{224 L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2}$.

What we did here was to set T just large enough so that the minimizer α is the same with or without the constraint.

Denoting $\tilde{W} = W_0(T^2 m^2 n^2 C_3) = \tilde{O}(1)$, we get

$$\begin{aligned}
F_{T+1} &\leq \frac{(F_1+\sigma^2/L_{2,\infty})\tilde{W}}{T^2 m^2 n^2 C_3} + \frac{112 L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2\tilde{W}^2}{T^2 m^2 n^2 \mu^3} \\
&\leq \frac{1}{T^2 m^2 n^2} \left(\frac{(F_1+\sigma^2/L_{2,\infty})\tilde{W}}{\tilde{C}_3} + \frac{112 L_{2,\infty}^2(\varsigma+\sigma)^2\tilde{A}^2\tilde{W}^2}{\mu^3} \right),
\end{aligned}$$

which shows rate the convergence rate in the P.L. case is $\tilde{O}((mnT)^{-2})$. □

D Experiment Details

Here we provide more extensive details on our empirical results. This includes background information on our experimental setup in the main paper (Appendix D.1), an additional simulation experiment on pre-training and fine-tuning Tiny GPT-2 (Appendix D.2), and an additional simulation experiment that investigates CD-GraB with different learning rates (Appendix D.3). Our source codes can be found here.

D.1 Additional details on setup for main paper experiments

D.1.1 Distributed experiments

We provide additional details on the experiments shown in Figure 3.

Hardware and software. We use a single machine with 128 GiB memory, 1 CPU, and 4 Nvidia GeForce 2080ti GPUs for the HMDA mortgage application, M4, and WikiText-2 tasks. We first discard the remainder $N \mod B$, and then randomly partition n to each worker. Our experiments are all implemented with the PyTorch library. We release our code suite at [REDACTED].

Datasets and models.

- **Logistic regression on mortgage application (NY 2017 subset)**: The US Home Mortgage Disclose Act (HMDA) makes available US national data regarding mortgage applications, which has recently been packaged up for easy ML research use [7]. We use the binary classification version of the task, which classifies features as either "grant loan" or "deny loan," for the New York (NY) 2017 subset of the dataset, which includes 244107 examples with 18 features. We model this problem using logistic regression, for which we first perform a random 80/20 train/test split on the raw dataset, and then we discard $N \mod B$ (B is the aggregated minibatch size) examples to ensure that each worker receives exactly n examples. We use 1 worker per GPU, and in total we have $m = 4$ workers, and use NCCL [33] as the distributed communication backend; $m = 4, n = 48816, d = 18, B = 16$. We report test accuracy as our evaluation metric.
- **LSTM on WikiText-2**: We follow the settings in Lu et al. [24] and train a 2-layer LSTM with an embedding size of 32 and dropout set to 0. We use backpropagation through time, for which we set the sequence length to 35. We also adopt the word-vector-classifier-weight-sharing strategy inspired by Inan et al. [19]. WikiText-2 [43] has 600 articles in the train set, with more than 2M tokens and 30K vocabulary; the validation and test sets each have 60 articles. We adapt our training script from PyTorch's official Word Language Modeling Github repository. We use 4 workers in total, with each GPU hosting 1 worker, and use NCCL as the distributed communication backend; $m = 4, n = 3728, d = 1081760, B = 16$. We report test perplexity as the evaluation metric, and we follow the HuggingFace's approach of computing perplexity as the exponentiated average negative log-likelihood of a sequence[16].
- **Autoregressive MLP on M4 Weekly Dataset**: We build a 3-layer autoregressive MLP with a hidden dimension of 64. We set input sequence length to be 20 and the output sequence length to be 6. M4 is a time series dataset composed of 100,000 time series for yearly, quarterly, monthly, weekly, daily and hourly data [25], which is drawn from a random sample of ForeDeCk database [42]. We use the weekly data in our experiment. We use 32 workers, where each of the 4 GPUs hosts 8 process workers. We use GLOO as the distributed communication backend. $m = 32, n = 3355, d = 5569, B = 32$. We report test symmetric mean absolute percentage error (SMAPE) as the evaluation metric. We follow the formula of SMAPE in [25] as follows:

$$\text{SMAPE} \triangleq \frac{2}{h} \sum_{t=n+1}^{n+h} \frac{|Y_t - \hat{Y}_t|}{|Y_t| + |\hat{Y}_t|} * 100\%$$

  where Y_t is the reference time series value at timestep t, $\hat{Y}_t$ is the forecast time series value at timestep t, and h is the forecasting horizon and n is the number of datapoints.

Hyperparameter optimization. For all tasks, we tune the learning rate α for D-RR first, and then use the selected learning rate for CD-GraB. Therefore, an performance improvement here implies we would have in-place substitution benefits via switching from D-RR to CD-GraB with identical learning rate and experiment setups. We use SGD with momentum as the optimizer for all tasks. The hyperparameters for each task are as follows:

[16] `https://huggingface.co/docs/transformers/perplexity`

- **Logistic regression on mortgage application (NY 2017 subset)**: $\alpha = $ 5e-3 $\in \{$1e-2, 5e-3, 1e-3$\}$, momentum: 0.9, weight decay: 0, B: 16.
- **LSTM on WikiText-2**: $\alpha = 5 \in \{5, 10\}$ and decays by 0.1 per 10 epochs, momentum: 0.9, weight decay: 0, B: 16.
- **Autoregressive MLP on Weekly M4 Dataset** $\alpha = $ 1e-3 $\in \{$1e-2, 1e-3, 1e-4$\}$, momentum: 0.9, weight decay: 0, B: 32.

D.1.2 Memory Overhead of CD-GraB in LSTM on WikiText-2 Task

We profile the CUDA memory usage for the LSTM on WikiText-2 Task with CD-GraB and D-RR to understand the memory overhead of both data permutation algorithms. This memory analysis is both task and implementation dependent, but still serves to illustrate the overarching point that CD-GraB's memory overhead is not so significant. The additional overhead comes from two sources for CD-GraB: communication and example sorting (Figure 8).

In more detail: In our LSTM experiment, each local worker will share its gradients with all other workers at every optimization step. To reduce the communication burden of CD-GraB, we make each local worker function as an order server.[17] The memory consumption of forward, backward, and optimizer states between CD-GraB and D-RR should be (at least approximately) identical. The model size of LSTM is roughly 4 MiB. We use 4 workers, and as each worker (functioning as an order server) needs to *all-gather* gradients, the memory overhead for *all-gather* communication is roughly tensor_size $\times$ # workers = 4 MiB $\times$ 4 = 16 MiB for CD-GraB(we observe 16.51 MiB in practice, Communication in Figure 8), while D-RR only needs to *all-reduce* the gradients (yielding no memory overhead; the communication buffer for *all-reduce* is reusing the same gradient tensor). The PairBalance algorithm (Algorithm 2) internally needs a model-sized accumulator as the running sum $\boldsymbol{r}$, and both computing inner product between $\boldsymbol{r}$ and $\boldsymbol{g}_1 - \boldsymbol{g}_2$ and updating $\boldsymbol{r}$ with $\boldsymbol{r} + s\boldsymbol{c}$ takes virtually no space with a memory-efficient implementation. Therefore, the memory consumption for PairBalance is still roughly 4 MiB (Data Sorter in Figure 8). .

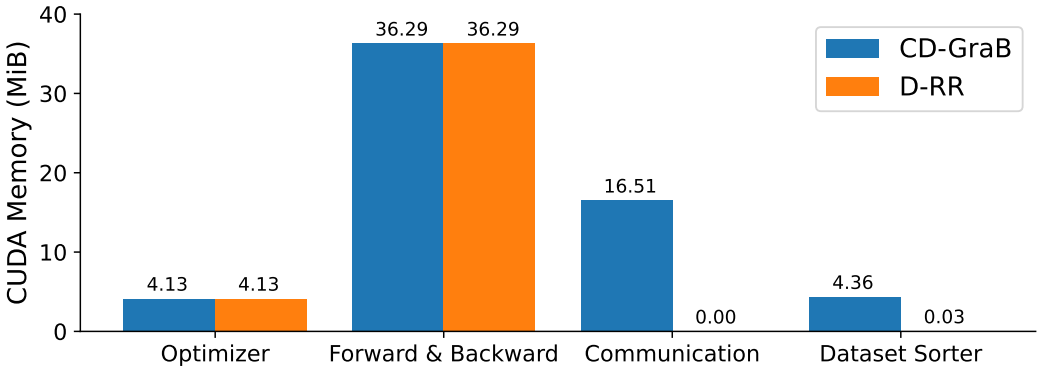


Figure 8: CUDA Memory Overhead of CD-GraB and D-RR in LSTM on WikiText-2 Task.

The main memory overhead of CD-GraB will be dominated by the communication buffer size on the order server side: the order server have to gather the gradient (differences) from all workers, and sequentially apply the PairBalance algorithm. This memory bottleneck would similarly be found in GraB as GraB also needs per-example gradients to perform Balance sequentially.

A future algorithmic improvement to the general gradient balancing framework would be finding a balancing algorithm that does not need per-example gradients to achieve comparable convergence guarantees. However, we still notice that PairBalance is more memory-efficient than Balance as Balance needs to store 3 model-sized tensors: 1 for the balancing accumulator, 1 for running-average gradients for last epoch, and 1 for the running-average for current epoch. In constrast, PairBalance only needs 1 model-sized tensor as the balancing accumulator.

[17] An ideal location for a dedicated order server is on a network node that has large input bandwidth and memory buffer to host all gradients while not blocking the normal optimization stages. Since we do not have enough computational resources to host a dedicated order server, we make each worker an order server.

D.1.3 Simulated ablation study using LeNet on CIFAR-10

In the experiment shown on Figure 4, we select the same learning rate, momentum, and weight decay as the LeNet experiment in Lu et al. [24]. We use 3 different random seeds to control 3 different initialization and the randomness in random reshuffling. The aggregated minibatch size B is 64 for all runs. We implement this ablation study by using 1 GPU with up to $m = 64$ workers (processes). As above, we discard $N \mod B$ examples and partition the remaining examples evenly on each worker.

$\alpha =$ 1e-3 $\in \{$1e-2, 5e-3, 1e-3, 5e-4, 1e-4$\}$, momentum: 0.9, weight decay: 1e-2, B: 64.

We do not implement this via distributed environment due to the fact that we do not have access to 64 GPUs, but expect the simulation results to be a good reflection of the results we would obtain in a multi-GPU setting.

Parallel herding bound. We further investigate the empirical parallel herding bounds (8) for the LeNet experiment for the different ordering methods. We plot the results in Figure 9. We observe that as the number of workers increases, the empirical parallel herding bounds of both **ID-GraB (Bal)** and **ID-GraB (PairBal)** also increase, and eventually exhibit little difference with D-RR. CD-GraB, in contrast, exhibits a consistently lower bound.

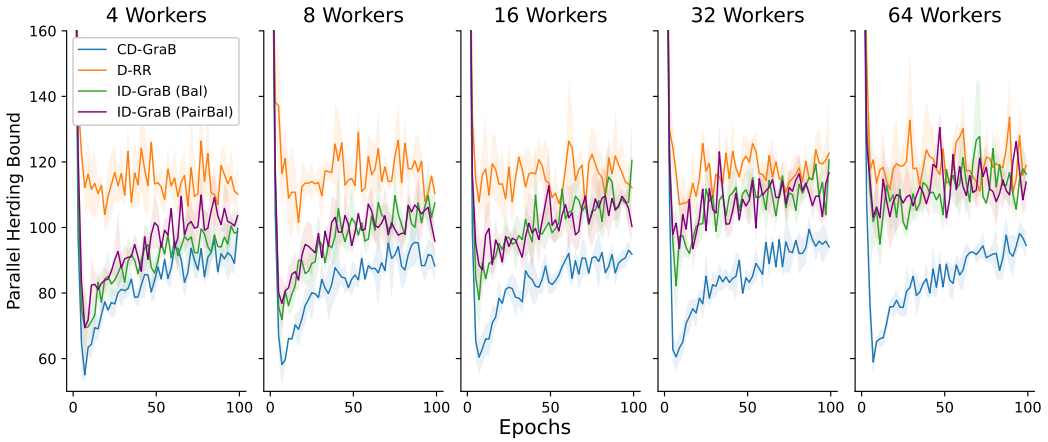


Figure 9: Empirical parallel herding bounds of gradients for each algorithm in LeNet experiment. We plot the mean as the curve and standard deviation across 3 random seeds.

For comparison, we also run a simulation experiment on synthetic data to investigate the behavior of the parallel herding bound. We include these below, in Figure 10.

We randomly initialize 1 million random vectors $\boldsymbol{z}_{i,j}$ from a uniform distribution between 0 and 1 with 16 dimensions as $\boldsymbol{z}_{i,j} \sim \mathsf{Unif}(0,1)^{16}$, and then we zero-center this set of 1 million vectors and normalize them to all have L_2 norm as 1. We then evenly partition this set of 1 million random vectors to $\{5,10,20,50,100\}$ workers and run each example ordering algorithm.

In Figure 10, we run CD-GraB, D-RR, **ID-GraB (Bal)**, **ID-GraB (PairBal)** on these random vectors, and compute the parallel herding bounds (8). From left to right in Figure 10, we observe that as the number of workers m increases, the parallel herding bound of **ID-GraB (Bal)**, **ID-GraB (PairBal)** becomes larger. This shows the importance of coordination when we have a large number of workers.

These results for random vectors cohere with our above results for LeNet on CIFAR-10.

D.2 An additional simulation experiment: pre-training and fine-tuning Tiny GPT-2

We perform an end-to-end simulation experiment involving pre-training and fine-tuning Tiny GPT-2 on WikiText-103, which we document below.

D.2.1 Pre-training

We adapt the training script from the HuggingFace's PyTorch casual language modeling code to train the GPT-2 architecture [35]. We set the maximum sequence length to 128 and token and positional

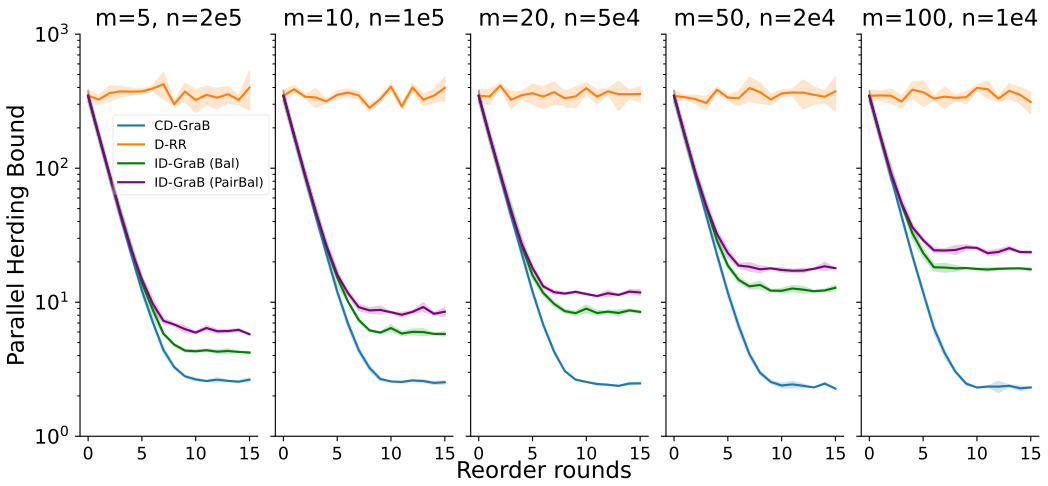


Figure 10: Parallel herding bounds for different example ordering algorithms on N=1 million random vectors. We use 3 random seeds, plot the mean and standard deviation across each random seed as the shaded area.

embedding dimension to 128; use 2 hidden layers in the transformer encoder and 2 attention heads; and disable dropout. This model configuration corresponds to the following Python code snippet:

```
from transformers import GPT2Config, GPT2LMHeadModel, GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
config = GPT2Config.from_pretrained('gpt2')
config.n_embd = 128
config.n_ctx = 128
config.n_layer = 2
config.n_head = 2
config.n_positions = 128
config.summary_first_dropout = 0
config.attn_pdrop = 0
config.resid_pdrop = 0
model = GPT2LMHeadModel(config)
```

We train our Tiny GPT-2 model from scratch on WikiText-103 [43]. WikiText-103 is a standard language modeling benchmark that has 28,475 articles in the train set, and 60 for both the validation and test sets, with more than 100M tokens and 267K vocabulary inside the train set. We use the original GPT-2 tokenizer, and use maximum sequence length 128. We note that this is much smaller than the default maximum sequence length for GPT-2, which is 1024, which was too large to use given our computational budget. Nevertheless, 128 is still a reasonable sequence length for the initial phrase of pre-training; BERT uses a sequence length of 128 for the first 90% of pre-training steps to speedup the experiment [9]. We tune the learning rate for D-RR with the grid { 5e-3, 1e-3, 5e-4, 1e-4 } (the final learning rate is 5e-4), and use AdamW optimizer [21]. We use 3 random seeds. Before the training, we simulate 64 workers, and similarly divide the training dataset evenly across them by discarding $N \mod B$ examples. Our hyperparameter optimization space is listed below:

Pretraining Hyperparameters. α=5e-4 $\in$ {5e-3, 1e-3, 5e-4, 1e-4}, weight decay: 1e-4, B: 64.

We document convergence for pre-training in Figure 11, and use test perplexity as our evaluation metric.

D.2.2 Fine-tuning

We then fine-tune the pre-trained Tiny GPT-2 model on downstream tasks. For each task, we load the pre-trained foundation model weights obtained at the end of 30 epochs of each example ordering algorithm after pretraining, and use the same example ordering algorithm to perform supervised fine-tuning. We focus on the largest 4 GLUE tasks [44]: MNLI, QQP, QNLI, and SST2. We tune the learning rate for D-RR with the AdamW optimizer, and for each run we report the best validation

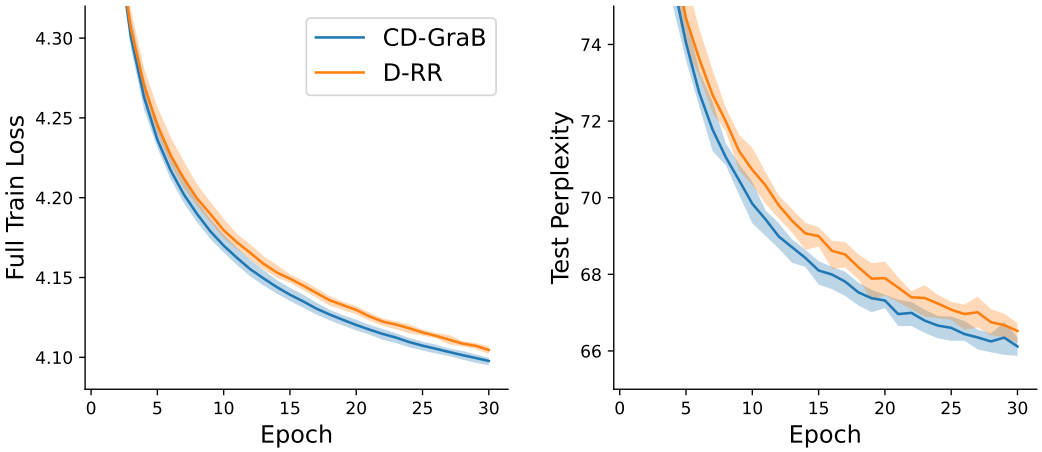


Figure 11: Pre-training Tiny GPT-2 on WikiText-103 from scratch: Convergence for CD-GraB and D-RR with $m = 64$ workers. The aggregated minibatch size per update is 64. We use 3 random seeds, and plot the mean vand standard deviation.

accuracy. We then take an average results of each run and summarize them in Table 1. Our training script is adapted from the HuggingFace's PyTorch GLUE fine-tuning example codes.

Fine-Tuning Hyperparameters

- **MNLI** α=5e-4$\in$ {5e-3, 1e-3, 5e-4, 1e-4}, Weight decay: 1e-4, B: 32, epochs: 10, linear learning rate scheduler
- **QQP** α=5e-4$\in$ {5e-3, 1e-3, 5e-4, 1e-4}, Weight decay: 1e-4, B: 32, epochs: 10, linear learning rate scheduler
- **QNLI** α=5e-4$\in$ {5e-3, 1e-3, 5e-4, 1e-4}, Weight decay: 1e-4, B: 32, epochs: 10, linear learning rate scheduler
- **SST2** α=5e-4$\in$ {5e-3, 1e-3, 5e-4, 1e-4}, Weight decay: 1e-4, B: 32, epochs: 10, linear learning rate scheduler

	MNLI (Matched)	**MNLI (Mismatched)**	**QQP**	**QNLI**	**SST2**
CD-GraB	65.91 $\pm$ 0.46 %	64.36 $\pm$ 2.03 %	82.25 $\pm$ 0.21 %	62.11 $\pm$ 0.70 %	82.65 $\pm$ 0.39 %
D-RR	65.42 $\pm$ 0.36 %	63.93 $\pm$ 1.63 %	81.74 $\pm$ 0.33 %	61.87 $\pm$ 0.67 %	82.68 $\pm$ 0.57 %

Table 1: GLUE fine-tuning datasets: Validation accuracy of CD-GraB in comparison to D-RR, reporting mean and standard deviation of best results for each run. There are 3 runs for each example ordering algorithm.

We include these fine-tuning results in part to support our claim in Section 6 that CD-GraB exhibits its benefits more clearly when there are more training epochs. Our pre-training results suggest that CD-GraB would confer benefits to pre-training large models over multiple epochs; however, CD-GraB will not necessarily be useful for short runs of fine-tuning (as indicated in Table 1, for which the results for both ordering algorithms are effectively identical).

D.3 Ablation simulation study: The impact of learning rate α

In the experiment shown on Figure 12, we select the same momentum and weight decay as the LeNet experiment for 3 random seeds as in Appendix D.1.3. The aggregated minibatch size is still 64 for all runs, and we use 64 workers.

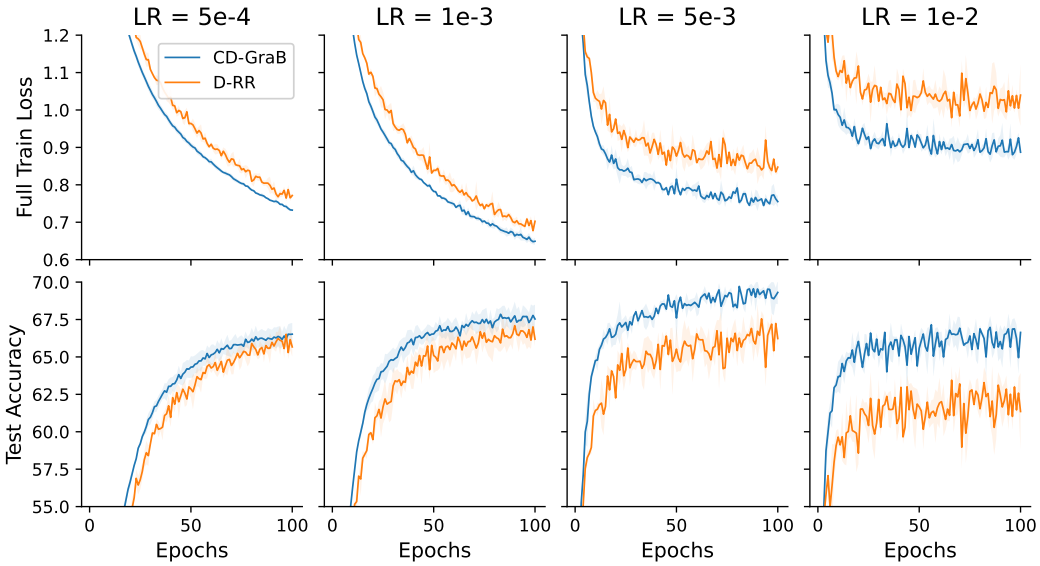


Figure 12: Convergence for CD-GraB, D-RR training LeNet on CIFAR-10, with $m = 64$ workers. The aggregated minibatch size per update is 64. We use 3 random seeds, and plot the mean values across random seeds as the curve, the standard deviation as the shaded area.

We find that when we increase the learning rate from 1e-3 to 1e-2, CD-GraB still maintains relatively better performance than D-RR. The best learning rate for D-RR is 1e-3, in terms of achieving the best test accuracy. We did not tune the learning rate for CD-GraB, and we expect that it is possible to use a higher learning rate and still maintain better empirical performance than D-RR and even faster convergence. We defer such empirical investigations to future work. Altogether, these preliminary empirical results confirm that it is possible to use higher learning rate for CD-GraB, given that online PairBalance does not need to use a stale mean (Section 3.2), which would make larger learning rates perform poorly.