# SOLON: Communication-efficient Byzantine-resilient Distributed Training via Redundant Gradients

Lingjiao Chen[1], Leshang Chen[2], Hongyi Wang[3], Edgar Dobriban[4], and Susan B. Davidson[2]
[1]*Dept. of Computer Sciences, Stanford Univ.,* [2]*Dept. of Computer and Information Science, Univ. of Pennsylvania*
[3]*Dept. of Computer Sciences, UW–Madison,* [4]*Statistics Dept., Wharton School, Univ. of Pennsylvania*
lchen362@wisc.edu, chenleshang@gmail.com, hwang595@wisc.edu, dobriban@wharton.upenn.edu, susan@seas.upenn.edu

***Abstract*—There has been a growing need to provide Byzantine-resilience in distributed model training. Existing robust distributed learning algorithms focus on developing sophisticated robust aggregators at the parameter servers, but pay less attention to balancing the communication cost and robustness. In this paper, we propose SOLON, an algorithmic framework that exploits gradient redundancy to provide communication efficiency and Byzantine robustness simultaneously. Our theoretical analysis shows a fundamental trade-off among computational load, communication cost, and Byzantine robustness. We also develop a concrete algorithm to achieve the optimal trade-off, borrowing ideas from coding theory and sparse recovery. Empirical experiments on various datasets demonstrate that SOLON provides significant speedups over existing methods to achieve the same accuracy, *e.g.*, over 10× faster than BULYAN and 80% faster than DRACO. We also show that carefully designed Byzantine attacks break SIGNUM and BULYAN, but do not affect the successful convergence of SOLON.**

## I. INTRODUCTION

The growing size of datasets and machine learning models has led to many developments in distributed training using stochastic optimization [9]–[12], [26]. One of the most widely used settings is the *parameter server (PS) model* [17], [20], [21], where the gradient computation is partitioned among all nodes, typically using stochastic gradient descent (SGD) or its variants. A central PS then aggregates the calculated gradients from all compute nodes to update the global model.

However, scaling PS models to large clusters introduces two challenges: guarding against *Byzantine attacks* and managing *communication overhead*. Byzantine attacks include erroneous gradients sent from unreliable compute nodes due to power outages, hardware or software errors, as well as malicious attacks. The communication overhead of sending gradients to the PS in large clusters can also be extremely high, potentially dominating the training time [1], [5], [9], [31], [32], since the number of gradients sent is linear in the number of nodes.

Although recent work has studied the problem of Byzantine attacks under the PS model [2], [14], [33], the communication overhead remains prohibitive. For example, [4], [6], [8], [35] use *robust aggregators* at the PS to mitigate unreliable gradients, while [5], [28] introduces *algorithmic redundancy* to detect and remove Byzantine nodes. Robust aggregators are computationally expensive due to their super-linear (often quadratic) dependence on the number of nodes. They also

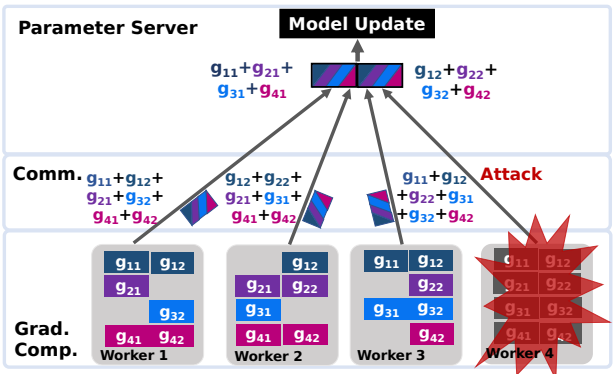



Figure 1. Example of SOLON, where the cluster consists of four compute nodes. There are four two-dimensional gradient vectors, $g_1, g_2, g_3, g_4$, where gradients $g_i = (g_{i1}, g_{i2})$ are in different colors. To tolerate Byzantine workers, each node computes four gradients. However, instead of sending a two-dimensional vector to the PS, each node only sends a one-dimensional linear combination of the elements in the local gradients. For instance, worker 1 computes and sends the scalar $g_{11} + g_{12} + g_{21} + g_{32} + g_{41} + g_{42}$. Here, the redundancy ratio $r = 4$ and the compression ratio $r_c = 2$. The PS uses a decoding scheme presented in Section II. Therefore SOLON can recover the correct gradient with one Byzantine worker ($4 \geq 2 \times 1 + 2$).

often have limited convergence guarantees under Byzantine attacks, e.g., only establishing convergence in the limit, or only guaranteeing the output of the aggregator has a positive inner product with the true gradient. They often require strong bounds on the dimension of the model. Although algorithmic redundancy or coding-theoretic [5], [28] approaches offer strong convergence guarantees, they have high communication overhead. Thus, it remains an open question to simultaneously provide ***Byzantine-resilience with strong guarantees and communication efficiency*** for distributed learning.

To address this problem, we propose SOLON, a distributed training framework leveraging algorithmic redundancy to protect against Byzantine attacks, and ideas from sparse recovery [25] and coding theory [28] to reduce communication overhead. The approach is as follows: In a Byzantine-free PS model with $P$ compute nodes and $B$ gradients to be computed, at each iteration each of the $P$ nodes computes $B/P$ gradients, and sends them to the PS. In SOLON, gradients are computed redundantly to tolerate Byzantine failures; each node computes $rB/P$ gradients, incurring a *computational redundancy ratio*

of $r$. To reduce communication overhead, compressed gradients are sent to the PS. For $d$-dimensional gradients, each node only sends a $d/r_c$-dimensional vector to the PS, where $r_c$ is the *compression ratio*.

We show that under worst-case adversarial conditions where the adversarial nodes have access to the complete data and gradients, and can send arbitrary results to the PS, there is a fundamental trade-off among Byzantine-resilience, communication overhead and computation cost. To tolerate $s$ Byzantine nodes, the redundancy ratio must satisfy $r \geq 2s + r_c$. We provide a concrete encoding and decoding technique for SOLON based on Vandermonde matrices and building on Prony's method from signal processing [25], [34] that matches the optimal condition $r = 2s + r_c$. An example is shown in Figure 1.

We implemented SOLON in Pytorch and conducted extensive experiments on a large cluster. Our results show that SOLON can provide significant speedups across various ML models and datasets over existing methods, such as BULYAN [14] and DRACO [5], shown in Figure 1(b) using ResNet-18 [15] on CIFAR-10 [19] under a reverse gradient attack. In addition, SOLON successfully defends against strong Byzantine attacks such as "A little is enough" (ALIE) [2], on which some methods such as BULYAN and SIGNUM [3] fail to converge or result in significant accuracy loss, see Section III.

***Our contributions include:***

1) SOLON, a distributed training framework that exploits algorithmic redundancy to simultaneously provide Byzantine-resilience and communication efficiency.
2) Optimal trade-offs between Byzantine resilience, communication overhead, and computational cost.
3) A concrete encoding and decoding mechanism, which is provably efficient and achieves this optimal trade-off.
4) Extensive experiments which show that SOLON exhibits significant speedups as well as strong Byzantine-resilience over previous approaches.

The SOLON framework can be used for any distributed algorithm which requires the sum of multiple functions, including gradient descent, SVRG [18], coordinate descent, and projected or accelerated versions of these algorithms. However, in this paper, we focus on mini-batch SGD.

## II. SOLON: COMMUNICATION EFFICIENT ROBUST DISTRIBUTED TRAINING VIA ALGORITHMIC REDUNDANCY

We present an overview for SOLON in this section. The proofs are left to a technical report in progress.

### A. Preliminaries

*a) Basic notations.:* For a matrix $\mathbf{A}$, let $\mathbf{A}_{i,j}$, $\mathbf{A}_{i,\cdot}$, and $\mathbf{A}_{\cdot,j}$ denote entries, rows, and columns, respectively. More generally, $\mathbf{A}_{S,T}$ is the submatrix of $\mathbf{A}$ with rows indexed by $S$ and columns indexed by $T$. The Hadamard, or elementwise, product $\mathbf{A} \odot \mathbf{B}$ of two matrices of the same size has entries $(\mathbf{A} \odot \mathbf{B})_{i,j} = \mathbf{A}_{i,j}\mathbf{B}_{i,j}$. Let $m$ be the dimension of the data, $n$ be the size of the training set, and $\mathbf{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, m$ be the data points. Let $\ell(\cdot;\cdot)$ be the loss function, $d$ be the model dimension, and $\mathbf{w} \in \mathbb{R}^d$ be the model parameters. Let $\mathbf{1}_m$ and $\mathbf{1}_{n\times m}$ be the $m \times 1$ vector, and $n \times m$ matrix, of all ones, respectively. Similarly, let $\mathbf{0}_m, \mathbf{0}_{n\times m}$ contain zeros. The empirical risk minimization (ERM) [29], [30] objective is: $\min_{\mathbf{w}} n^{-1} \sum_{i=1}^{n} \ell(\mathbf{w}; \mathbf{x}_i)$. The most common current approach is to use first-order stochastic optimization to solve this, in particular mini-batch stochastic gradient descent (SGD). Starting at an initial point $\mathbf{w}_0$, we iterate $\mathbf{w}_k = \mathbf{w}_{k-1} - \gamma/|S_k| \sum_{i\in S_k} \nabla\ell_{\mathbf{w}}(\mathbf{w}_{k-1}; \mathbf{x}_i)$, where $S_k \subseteq \{1, \ldots, n\}$ is a random subset of size $B$ and $\gamma > 0$ is the learning rate. We relabel $S_k$ to $\{1, \ldots, B\}$ and denote $\nabla\ell(\mathbf{w}_{k-1}; \mathbf{x}_i)$ by $\mathbf{g}_i$.

*b) Distributed learning.:* We aim to compute $\mathbf{g} = \sum_{i=1}^{B} \mathbf{g}_i$ in a distributed, *adversary-resistant*, and *communication efficient* manner. We consider a distributed training model where gradient computations are partitioned across $P$ compute nodes at each iteration. These operate on a potentially reduced dimension $d_c$ for communication efficiency, and we let the gradient *compression ratio* be $r_c \triangleq d/d_c$. After computing and summing up their assigned gradients, each node sends their answer back to the parameter server (PS). This sums them and updates the model. By applying SOLON, we reduce the communication complexity for sending gradients to server from $\mathcal{O}(Pd)$ to $\mathcal{O}(Pd_c)$. The broadcast phase of sending aggregated gradients from server to compute nodes takes $\mathcal{O}(\log(P)d)$, which is not the major overhead. We assume that at most $s$ compute nodes are unreliable, Byzantine, or adversarial, and can send to the PS an arbitrary update. We consider the strongest possible adversaries: with infinite computational power, knowing the entire data set, the training algorithm, any defenses present in the system, and able to collaborate.

### B. Framework

SOLON is defined by the tuple, or *mechanism*, $(\mathbf{A}, E, D)$, where $\mathbf{A}$ is an allocation matrix specifying how to assign gradients to nodes, $E$ are encoding functions determining how each compute node should locally encode its gradients, and $D$ is a decoding function determining how the PS should decode the output of the nodes. As an example, in Figure 1(a), $A$ corresponds to the gradient computation assignment of the compute nodes, $E$ corresponds to the summation of the gradients by each node, and $D$ refers to the decoding phase at the PS. We generalize the scheme in Figure 1 to $P$ compute nodes and $B$ gradients.

**Allocation matrix, A.** At each iteration of the training process, we assign the $B$ gradients to the $P$ compute nodes using a $P \times B$ *allocation matrix* $\mathbf{A}$, where $\mathbf{A}_{j,k}$ is equal to unity ("1") if node $j$ is assigned to the $k$th gradient $\mathbf{g}_k$, and zero ("0") otherwise. The support of $\mathbf{A}_{j,\cdot}$, denoted $\text{supp}(\mathbf{A}_{j,\cdot})$, is the set of indices of gradients evaluated by the $j$th node. For simplicity, we will assume $B = P$. Let $\|\mathbf{A}\|_0$ be to the $L_0$ norm of a matrix, i.e., the number of nonzero entries. Following [5], we define the *redundancy ratio* of an allocation as the average number of gradients assigned to each compute node, or equivalently $r \triangleq \|\mathbf{A}\|_0/P$. We define the $d \times P$ matrix $\mathbf{G}$ with gradients as its columns: $\mathbf{G} \triangleq [\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_P]$. The $j$th node first picks out its assigned gradients using the allocation matrix $\mathbf{A}$, computing a $d \times P$ gradient matrix $\mathbf{Y}_j \triangleq (\mathbf{1}_d \mathbf{A}_{j,\cdot}) \odot \mathbf{G}$.

The columns of this matrix are $\mathbf{g}_k$ if the $k$th gradient $\mathbf{g}_k$ is allocated to the $j$th compute node, *i.e.*, $\mathbf{A}_{j,k} \neq 0$, and zero otherwise.

**Encoding Functions,** $E$**.** The $j$th compute node is equipped with an encoding function $E_j$ that maps the $d \times P$ matrix $\mathbf{Y}_j$ of its assigned gradients to a $d_c$-dimensional vector. The $j$th compute node computes and sends $\mathbf{z}_j \triangleq E_j(\mathbf{Y}_j)$ to the PS. If the $j$th node is adversarial, then it instead sends $\mathbf{z}_j + \mathbf{n}_j$ to the PS, where $\mathbf{n}_j$ is an $d_c$-dimensional *Byzantine* vector. We let $E = \{E_1, E_2, \cdots, E_P\}$ be the set of local encoding functions.

**Decoding Function,** $D$. The $d_c \times P$ matrix $\mathbf{Z} = \mathbf{Z}^{\mathbf{A},E,\mathbf{G}} \triangleq [\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_P]$ contains all outputs of the nodes. The $d_c \times P$ matrix $\mathbf{N} \triangleq [\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_P]$ contains all Byzantine vectors, with at most $s$ non-zero columns. Then, the PS receives a $d_c \times P$ matrix $\mathbf{R} \triangleq \mathbf{Z} + \mathbf{N}$, and computes a $d$-dimensional vector $\mathbf{u} \triangleq D(\mathbf{R})$ using a decoding function $D$. We require that the algorithm at the PS recovers the $d$-dimensional sum of gradients, $\mathbf{G}\mathbf{1}_P$:

**Definition 1.** SOLON *with* $(\mathbf{A}, E, D)$ *can tolerate* $s$ *adversarial nodes, if for any* $\mathbf{N} = [\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_P]$ *such that* $|\{j : \mathbf{n}_j \neq 0\}| \leq s$*, we have* $D(\mathbf{Z} + \mathbf{N}) = \mathbf{G}\mathbf{1}_P$.

If we defend against the Byzantine attack, then the model update at each iteration is identical to the adversary-free setting. This implies that convergence guarantees for the adversary-free case transfer to the adversarial case.

### C. Encoding and decoding functions

What are the fundamental limits of the above allocation, encoding, and decoding schemes, in particular of the redundancy ratio used in allocation and the compression ratio used in encoding? Perhaps surprisingly, *the redundancy ratio does not depend on the compression ratio*. The encoded gradients at each compute node can be arbitrarily compressed without affecting the our ability to tolerate Byzantine attacks.

**Theorem 1.** *If there is a mechanism* $(\mathbf{A}, E, D)$ *of gradient allocation, encoding, and decoding with redundancy ratio* $r$ *tolerating* $s$ *adversarial nodes with a compression ratio* $r_c$ *of unity, then there is a mechanism* $(\mathbf{A}', E', D')$ *with redundancy ratio* $r$ *tolerating* $s$ *adversarial nodes for any compression ratio* $r_c > 0$.

However $(\mathbf{A}', E', D')$ is in a sense pathological and it is unclear if it can reduce the the number of bits communicated. Therefore, we seek classes of *regular* encoder and decoder functions $E, D$, to reduce communication cost.

**Definition 2.** *A set of encoding functions* $E$ *is called* regular *if each output element of each function* $E_j$ *is a function of linear combinations of columns of the input. Formally, if* $E_{j,v}$ *is the* $v$*th element of* $E_j$*, then* $(\mathbf{A}, E, D)$ *is regular if there exists a* $d \times P$ *matrix* $\mathbf{U}_{j,v}$ *and functions* $\hat{E}_{j,v}$*, such that* $E_{j,v}(\mathbf{Y}_j) = \hat{E}_{j,v}\left(\mathbf{1}_d^T \left(\mathbf{U}_{j,v} \odot \mathbf{Y}_j\right)\right)$.

*a) Redundancy Bound.:* We first study redundancy requirements for exact recovery of the sum of gradients with $s$ adversaries and compression ratio $r_c$.

**Theorem 2.** *A mechanism* $(\mathbf{A}, E, D)$ *of gradient allocation, regular encoding, and decoding with compression ratio* $r_c$ *tolerating* $s$ *adversarial nodes must have a redundancy ratio* $r \geq 2s + r_c$.

Thus, for any regular encoder, each gradient has to be replicated on average at least $2s + r_c$ times to defend against $s$ adversarial nodes with a communication compression ratio of $r_c$. If a mechanism tolerates $s$ adversarial nodes with a communication compression ratio of $r_c$, by Theorem 2, each compute node encodes at least $(2s + r_c)$ $d$-dimensional vectors on average. If the encoding has linear time complexity, then each encoder requires $O((2s + r_c)d)$ operations in the worst case. If the decoder $D$ has linear time complexity, then it requires at most $O(Pd_c)$ operations in the worst case, as it needs to use the $d$-dimensional input from all $P$ compute nodes. This gives a computational cost of $O(Pd_c)$, which is less than the bound $\mathcal{O}(Pd)$ for the repetition code in [5].

### D. Optimal Coding Schemes

Can we design a tuple $(\mathbf{A}, E, D)$ that has redundancy ratio $r = 2s + r_c$? We give a positive answer by presenting a class of coding approaches that match the above bounds.

*a) Linear Block Code:* Suppose $2s + r_c$ divides $P$. Divide all compute nodes into $\frac{P}{2s+r_c}$. We assign each node in the same group to compute the same gradients ("block"). Summing all computed gradients as a single long vector, each compute node computes some linear combinations of elements in this vector and then send them to the PS ("linear"). The PS obtains the desired gradient summation within each group by solving a few systems of linear equations.

Formally, the linear block code $(\mathbf{A}^{LBC}, E^{LBC}, D^{LBC})$ is defined as follows. The assignment matrix $\mathbf{A}^{LBC}$ is given by

$$\mathbf{A}^{LBC} = \begin{bmatrix} \mathbf{1}_{r\times r} & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} \\ \mathbf{0}_{r\times r} & \mathbf{1}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{1}_{r\times r} \end{bmatrix}.$$

The $j$th compute node first computes all its allocated gradients $\mathbf{Y}_j^{LBC} = \left(\mathbf{1}_d \mathbf{A}_{j,\cdot}^{LBC}\right) \odot \mathbf{G}$. Its encoder function first computes the summation of all the allocated gradients $\mathbf{Y}_j^{LBC}\mathbf{1}_P$, and then computes $\mathbf{W}_j \mathbf{Y}_j^{LBC}\mathbf{1}_P$ where $\mathbf{W}_j$ is some $d_c \times d$ matrix. That is, $E_j^{LBC}(\mathbf{Y}_j^{LBC}) = \mathbf{W}_j \mathbf{Y}_j^{LBC}\mathbf{1}_P$. It then sends $\mathbf{z}_j = E_j^{Rep}(\mathbf{Y}_j^{Rep})$ to the PS.

Next, we present the decoder function, summarized in Algorithm 1. The decoder function first partitions the received updates into $\frac{P}{r}$ groups, within each of which all nodes are required to compute the same gradients (corresponding to the encoder function). For each group, a block decoder function $\psi(\cdot, \cdot)$ recovers the sum of all gradients computed in this group.

We then discuss how $\mathbf{W}_j$ is constructed. Given distinct $w_1, w_2, \cdots, w_P$ where $w_j \neq 0, \forall j$, $\mathbf{W}_j$ is constructed by $\mathbf{W}_j \triangleq \mathbf{I}_{d_c} \otimes [1, w_j, w_j^2, \cdots, w_j^{r_c-1}]$. The adversarial node index locating function $\phi(\cdot)$ works as follows. Given the $d_c \times r$ matrix $\mathbf{R}_j^{LBC}$ received from the compute nodes, we first generate a $1 \times d_c$ random vector $\mathbf{f} \sim \mathcal{N}(\mathbf{1}_{1\times d_c}, \mathbf{I}_d)$, and then compute $\mathbf{r}_{j,c} \triangleq \mathbf{f}\mathbf{R}_j^{LBC}$. Next, we obtain a $r$-dimensional

**Algorithm 1** Decoder Function $D^{LBC}$.

**Input** : Received $d_c \times P$ matrix $\mathbf{R}^{LBC}$
**Output** : Desired gradient summation $\mathbf{u}^{LBC}$

Let $\mathbf{R}^{LBC} = \left[\mathbf{R}_1^{LBC}, \mathbf{R}_2^{LBC}, \cdots, \mathbf{R}_{\frac{P}{r}}^{LBC}\right]$, where each $\mathbf{R}_j^{LBC}$ is a $d_c \times r$ matrix
**for** $j = 1$ **to** $\frac{P}{r}$ **do**
  $V = \phi(\mathbf{R}_j^{LBC}, j)$ // Locate the adversarial nodes
  $U = \{1, 2, \cdots, r\} - V$ // Non-adversarial nodes
  $\mathbf{u}_j^{LBC} = \psi(\mathbf{R}_j^{LBC}, j, U)$ // Decode each block using the non-adversarial nodes
**end**
Compute and return $\mathbf{u}^{LBC} = \sum_{i=1}^{\frac{P}{r}} \mathbf{u}_i^{LBC}$

vector $\mathbf{a} = [a_1, a_2, \cdots, a_r]$ by solving the linear system $\left[\hat{\mathbf{W}}_{j,rc+s-1}, \quad -\hat{\mathbf{W}}_{j,s-1} \odot \left(\mathbf{r}_{j,c}^T \mathbf{1}_s^T\right)\right] \mathbf{a} = \mathbf{r}_{j,c} \odot \left[\hat{\mathbf{W}}_{j,s}\right]_{\cdot,s}$,

$$\hat{\mathbf{W}}_{j,v} \triangleq \begin{bmatrix} 1 & w_{(j-1)r+1} & w_{(j-1)r+1}^2 & \cdots & w_{(j-1)r+1}^v \\ 1 & w_{(j-1)r+2} & w_{(j-1)r+2}^2 & \cdots & w_{(j-1)r+2}^v \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & w_{jr} & w_{jr}^2 & \cdots & w_{jr}^v \end{bmatrix}.$$

Finally compute $P_j(w) \triangleq (\sum_{i=0}^{r_c+s-1} a_{i+1} w^i)/(w^s + \sum_{i=0}^{s-1} a_{i+r_c+s+1} w^i)$, and return $V = \{i | P_j(w_{(r-1)j+i}) \neq [\mathbf{r}_{j,c}]_i\}$. The decoding function $\psi(\cdot,\cdot)$ works as follows. Given the non-adversarial node index $U$, it computes and returns $vec\left(\left[\mathbf{R}_j^{LBC}\right]_{\cdot,U} \left[\left[\hat{W}_{j,r_c-1}\right]_{U,\cdot}^{-1}\right]^T\right)$. The following lemma ensures that the Byzantine nodes are correctly found.

**Lemma 3.** *Suppose* $|\{j : \|\mathbf{n}_j\|_0 \neq 0\}| \leq s$ *and* $r \geq r_c + 2s$. *Then* $\phi(\mathbf{R}_j^{LBC}, j) = \{i : \|\mathbf{n}_{j(r-1)+i}\|_0 \neq 0\}$ *w. probability 1.*

The next Lemma demonstrates that within each group the gradient is correctly recovered.

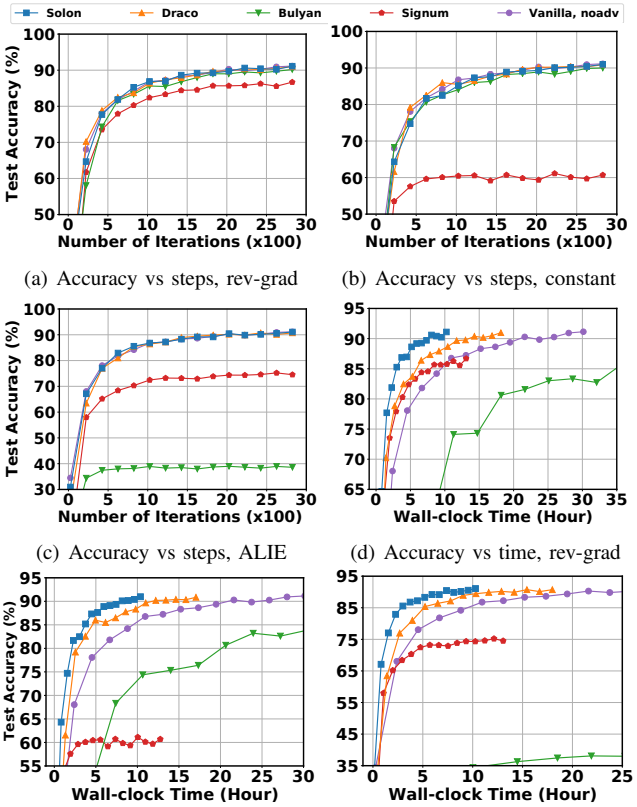
**Lemma 4.** *If* $U$ *only contains* $\geq r - s$ *non-adversarial nodes, then with probability 1,* $\mathbf{u}_j^{LBC} = \sum_{k=(j-1)r+1}^{jr} \mathbf{y}_k$.

Combing the above two results, we can show that the linear block code can tolerate any $s$ adversaries and also achieves redundancy ratio, compression ratio, and has linear-time encoding and decoding.

**Theorem 5.** *The linear block code* $(\mathbf{A}^{LBC}, E^{LBC}, D^{LBC})$ *tolerates any* $s$ *adversaries with probability 1 and achieves the redundancy ratio bound. For* $d \gg P$, *its encoding and decoding achieve linear-time computational complexity.*

## III. Experiments

We present an empirical study on SOLON compared with several existing methods including DRACO, BULYAN, and SIGNUM. Across diverse ML models trained on real world datasets, we have found 1) that SOLON results in significant speedups over existing methods, including $11\times$ faster than BULYAN and $80\%$ faster than DRACO while reaching the same accuracy, and 2) that SOLON consistently leads to successful convergence for all Byzantine attacks considered, while previous approaches may fail on different attacks. We leave more detailed experiments in an on-going technical report.

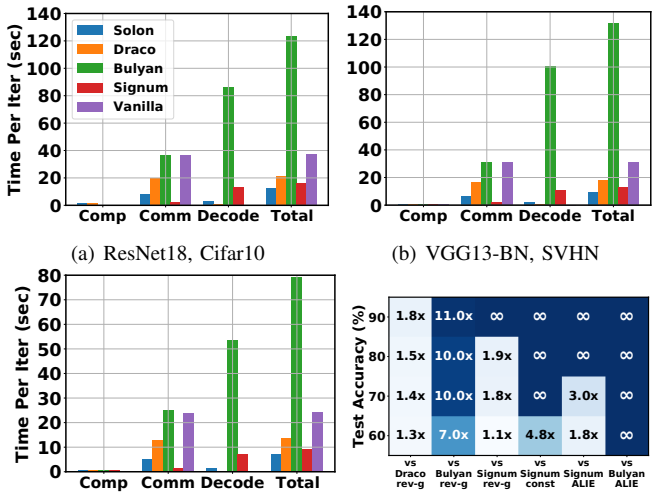



(a) Accuracy vs steps, rev-grad (b) Accuracy vs steps, constant (c) Accuracy vs steps, ALIE (d) Accuracy vs time, rev-grad (e) Accuracy vs time, constant (f) Accuracy vs time, ALIE

Figure 2. End to end convergence performance of SOLON and other baselines on ResNet-18 and CIFAR-10. (a)-(c): Comparison of test accuracy vs. the number of iterations between SOLON $r_c = 10$ and other methods under different attacks. (d)-(f): Test accuracy vs. running time of SOLON and other methods. Vanilla SGD simply averages gradients received on PS and is tested without adversary. Accuracy may fluctuate occasionally due to randomness and lr adjustments.

*a) Experimental setup:* We implement SOLON in PyTorch [24] with MPI [7] as the communication library. The experiments were conducted on a cluster of 50 real c220g5 machines from Cloudlab [13] with 100 virtual machines and a 1 Gbps network speed. We trained three large scale models, namely, ResNet-18 [15] on CIFAR-10 [19], VGG13-BN [27] on SVHN [23], and a two-layer stacked LSTM [16] on Wikitext-2 [22], respectively. For comparison with SOLON, we also evaluate two robust aggregator-based approches, BULYAN, SIGNUM, and DRACO. SOLON splits the virtual machines evenly into 5 groups, each with 20 redundant machines.

*b) Attacks:* We use three different attacks: reverse gradient, constant, and ALIE. In the reverse gradient attack (rev-grad), Byzantine nodes always send $\kappa$ times the true gradient to the PS. In the constant attack, Byzantine nodes always send a constant multiple $c$ of the all-ones vector. In the experiments shown, $\kappa = -100$ and $c = -100$. In ALIE, Byzantine nodes use local information to estimate the mean and variance of the gradients computed at the other nodes, and then manipulate




(a) ResNet18, Cifar10 (b) VGG13-BN, SVHN

(c) LSTM, Wikitext-2 (d) Speedup of SOLON

Figure 3. (a)-(c) Time breakdown per iteration of SOLON $r_c = 10$ and all baseline methods over three models+datasets under rev-grad. The types of attack will not affect any of the times. Notice that even if SIGNUM is the fastest in time per iteration among those methods, and even faster than vanilla SGD, its accuracy is sacrificed and thus SOLON still has the best convergence performance. (d) The speed up ($\times$) of SOLON, $r_c = 10$ vs other baselines on converging to certain test accuracy level on ResNet-18+Cifar10 under selected attacks. Values are approximation. $\infty$ means the method does not converge.

the gradient as $\hat{\mu} + z \cdot \hat{\sigma}$ where $\hat{\mu}$ and $\hat{\sigma}$ are the estimated mean and standard deviation by Byzantine nodes and $z$ is an adjustable hyper-parameter. In experiments, we set $z = 1$. At each iteration, we randomly select $s = 5$ nodes as adversaries.

*c) End to end performance:* The end to end performance is shown in Figure 2. We first note that previous approaches may result in significant accuracy loss under certain attacks. For example, SIGNUM's accuracy is 30% worse than the Byzantine-free vanilla SGD under constant attack (Figure 2(b)), and ALIE attack leads a 50% accuracy drops for BULYAN (Figure 2(b)). Nevertheless, across different attacks, SOLON consistently converges and matches the accuracy performance of the vanilla SGD in a Byzantine-free environment.

Furthermore, SOLON provides significant runtime speedups over existing methods. As shown in Figure 2(d), SOLON converges faster than all the other baselines. Its runtime performance even outperforms the vanilla SGD in a Byzantine-free environment. This is primarily due to the communication compression technique used in SOLON. Figure 3(d) gives a quantitative result of the speedups achieved by SOLON. To achieve a 90% test accuracy, SOLON obtains a speedup of $1.8\times$ over DRACO and $11\times$ over BULYAN under the reverse gradient attack, while SIGNUM cannot reach 90% accuracy.

*d) Per iteration cost:* Next we dive into the per iteration cost of each approach, as shown in Figure 3(a)-(c). We note that BULYAN requires a significantly higher decoding time than all the other methods. This is probably because BULYAN deploys a computational expensive robust aggregator. Note that SOLON reduces the communication cost by slightly increasing the computation and decoding complexity compared to DRACO. Nevertheless, across all datasets and models considered in our experiments, SOLON attains the fastest per iteration runtime. This is because SOLON largely reduces the communication cost, which is the bottleneck in a large cluster, and the extra computation and decoding cost is relatively small.

## IV. CONCLUSION

In this paper, we propose SOLON, a distributed training framework that simultaneously resists Byzantine attack and reduces communication overhead via algorithmic redundancy. We show that there is a fundamental trade-off between Byzantine-resilience, communication cost, and computational cost. Extensive experiments show that SOLON provides significant speedups over existing methods, and consistently leads to successful convergence under different attacks.

## REFERENCES


[1] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, 2017, pp. 1707–1718.

[2] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019, pp. 8635–8645.

[3] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signSGD with majority vote is communication efficient and fault tolerant," in *ICLR*, 2019.

[4] P. Blanchard, R. Guerraoui, J. Stainer *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 119–129.

[5] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *International Conference on Machine Learning*, 2018, pp. 903–912.

[6] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[7] L. Dalcín, R. Paz, and M. Storti, "Mpi for python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108–1115, 2005.

[8] G. Damaskinos, E. El-Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault, "AGGREGATHOR: byzantine machine learning via robust gradient aggregation," in *MLSys*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., 2019.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*. Ieee, 2009, pp. 248–255.

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL*, J. Burstein, C. Doran, and T. Solorio, Eds., 2019.

[12] E. Dobriban and Y. Sheng, "Wonder: Weighted one-shot distributed ridge regression in high dimensions." *Journal of Machine Learning Research*, vol. 21, no. 66, pp. 1–52, 2020.

[13] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14.

[14] R. Guerraoui, S. Rouault *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, 2018, pp. 3521–3530.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[16] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, ser. Advances in Neural Information Processing Systems. MIT Press, 1996, p. 473–479.

[17] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous gpu/cpu clusters," in *OSDI*, 2020, pp. 463–479.

[18] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[19] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[20] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014, pp. 583–598.

[21] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *Big Learning NIPS Workshop*, vol. 6, 2013, p. 2.

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

[23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 01 2011.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library." in *Advances in Neural Information Processing Systems*, 2019.

[25] B. Recht. (2010) Prony's method. CS838 Topics in optimization: Convex geometry in high-dimensional data analysis, Lecture 6. [Online]. Available: http://pages.cs.wisc.edu/~brecht/cs838docs/Lecture06.pdf

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

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[28] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[29] V. Vapnik, "Principles of risk minimization for learning theory," in *Advances in Neural Information Processing Systems*, J. Moody, S. Hanson, and R. P. Lippmann, Eds., vol. 4. Morgan-Kaufmann, 1992.

[30] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[31] H. Wang, S. Sievert, S. Liu, Z. B. Charles, D. S. Papailiopoulos, and S. Wright, "ATOMO: communication-efficient learning via atomic sparsification," in *Advances in Neural Information Processing*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 9872–9883.

[32] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *arXiv preprint arXiv:1705.07878*, 2017.

[33] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 261–270.

[34] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Internation Conference on Machine Learning*, vol. 80. PMLR, 10–15 Jul 2018, pp. 5610–5619.

[35] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Internation Conference on Machine Learning*. PMLR, 2018, pp. 5650–5659.