Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy

Qian Yu*, **Songze Li***, **Netanel Raviv**†, **Seyed Mohammad Mousavi Kalan***, **Mahdi Soltanolkotabi***, **and A. Salman Avestimehr***

* Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA

† Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, USA

Abstract

We consider a scenario involving computations over a massive dataset stored distributedly across multiple workers, which is at the core of distributed learning algorithms. We propose *Lagrange Coded Computing* (LCC), a new framework to *simultaneously* provide (1) *resiliency* against stragglers that may prolong computations; (2) *security* against Byzantine (or malicious) workers that deliberately modify the computation for their benefit; and (3) (information-theoretic) *privacy* of the dataset amidst possible collusion of workers. LCC, which leverages the well-known Lagrange polynomial to create computation redundancy in a novel coded form across workers, can be applied to any computation scenario in which the function of interest is an *arbitrary multivariate polynomial* of the input dataset, hence covering many computations of interest in machine learning. LCC significantly generalizes prior works to go beyond linear computations. It also enables secure and private computing in distributed settings, improving the computation and communication efficiency of the state-of-the-art. Furthermore, we prove the optimality of LCC by showing that it achieves the optimal tradeoff between resiliency, security, and privacy. Finally, we show via experiments on Amazon EC2 that LCC speeds up the conventional uncoded implementation of distributed least-squares linear regression by up to $13.43\times$, and also achieves a $2.36\times$-$12.65\times$ speedup over the state-of-the-art straggler mitigation strategies.

1 Introduction

The massive size of modern datasets necessitates computational tasks to be performed distributedly, where data is dispersed among many servers operating in parallel [1]. As we "scale out" computations across many servers, several fundamental challenges arise. Cheap commodity hardware tends to vary greatly in computation time, and it has been demonstrated [2–4] that a small fraction of servers, referred to as *stragglers*, can be 5 to 8 times slower than average, creating significant delays in computations. Also, as we distribute computations across many servers, massive amounts data must be moved between them to execute the computational tasks, often over many iterations of a running algorithm, and this creates a substantial bandwidth bottleneck [5]. Distributed computing systems are also much more susceptible to adversarial servers, making security and privacy a major concern [6–8].

We consider a general scenario where computation is carried out distributively across several workers, and propose *Lagrange Coded Computing* (LCC), a new framework to simultaneously provide

1. *resiliency* against straggler workers that may prolong computations;
2. *security* against Byzantine (or malicious, adversarial) workers, with no computational restriction, that deliberately send erroneous data in order to affect the computation for their benefit; and
3. *(information-theoretic) privacy* of the dataset amidst possible collusion of workers.

LCC can be applied to any computation scenario where the function of interest is an *arbitrary multivariate polynomial* of the input dataset. This covers many computations of interest in machine

Workshop on Systems for ML and Open Source Software at NeurIPS 2018, Montréal, Canada.

learning, such as various gradient and loss-function computations and tensor algebraic operations (e.g., low-rank tensor approximation). The key idea of LCC is to encode the input using Lagrange polynomials, to create computational redundancy in a novel coded form across workers. This redundancy can then be exploited to provide resiliency, security, and privacy.

Specifically, as illustrated in Fig. 1, using a master-worker distributed computing architecture with N workers, the goal is to compute $f(X_i)$ for every X_i in a large dataset $X = (X_1, X_2, \ldots, X_K)$, where f is a given polynomial with degree $\deg f$. To do so, N coded versions of the input dataset, denoted by $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_N$ are created, and the workers then compute f over the coded data, *as if no coding is taking place*. For a given N and f, we say that the tuple (S, A, T) is *achievable* if there exists an encoding and decoding scheme that can complete the computations in the presence of up to S stragglers, up to A adversarial workers, whilst keeping the dataset private against sets of up to T colluding workers.

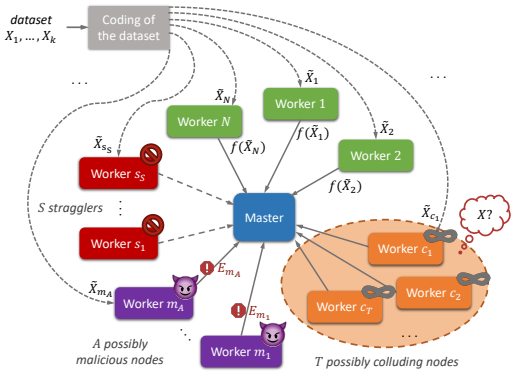

Figure 1: An overview of the problem considered in this paper, where the goal is to evaluate a *not necessarily linear* function f on a given dataset $X = (X_1, \ldots, X_K)$ using N workers. Each worker applies f on a possibly *coded* version of inputs (denoted by $\tilde{X}_i$'s). By carefully designing the coding strategy, the master can decode all the required results from a subset of workers, in the presence of *stragglers* (workers $s_1, ..., s_S$) and *Byzantine workers* (workers $m_1, ..., m_A$), while keeping the dataset private to *colluding workers* (workers $c_1, ..., c_T$).

Our main result is that by carefully encoding the dataset the proposed LCC achieves (S, A, T) if $(K + T - 1)\deg f + S + 2A + 1 \leq N$. The significance of this result is that by one additional worker (i.e., increasing N by 1) LCC can increase the resiliency to stragglers by 1 or increase the robustness to malicious servers by $1/2$, while maintaining the privacy constraint. Hence, this result essentially extends the well-known optimal scaling of error-correcting codes (i.e., adding one parity can provide robustness against one erasure or $1/2$ error in optimal maximum distance separable codes) to the distributed secure computing paradigm.

We prove the optimality of LCC by showing that it achieves the optimal tradeoff between resiliency, security, and privacy. This further extends the scaling law in coding theory, showing that similar to the resiliency and security requirement, it is fundamental that any additional worker increases data privacy against colluding workers by $1/\deg f$.

Finally, we specialize our theoretical guarantees of LCC in context of least-squares linear regression, an elemental learning task, and demonstrate its performance gain by optimally suppressing stragglers. Leveraging the algebraic structure of gradient computation, several strategies have been developed recently to exploit data and gradient coding for straggler mitigation in the training process (e.g. [9–13]). We implement LCC for regression on Amazon EC2 clusters, and empirically compare its performance with conventional uncoded approaches and two state-of-the-art straggler mitigation schemes: gradient coding (GC) [10, 14–16] and matrix-vector multiplication (MVM) based approaches [9, 11]. Our experiment shows that compared with the uncoded scheme, GC scheme, and MVM scheme, LCC improves the run-time by $6.79\times$-$13.43\times$, $2.36\times$-$4.29\times$, and $1.01\times$-$12.65\times$, respectively.

Related works. There has recently been a surge of interest on using coding theoretic approaches to alleviate key bottlenecks (e.g., stragglers, bandwidth, and security) in distributed machine learning applications (e.g., [10, 14, 15, 17–25]). The proposed LCC scheme significantly advances prior works in this area by 1) generalizing coded computing to arbitrary multivariate polynomial computations, which are of particular importance in learning applications; 2) extending the application of coded computing to secure and private computing; 3) reducing the computation/communication load in distributed computing (and distributed learning) by factors that scale with the problem size, without compromising security and privacy guarantees; and 4) enabling $2.36\times$-$12.65\times$ speedup over the state-of-the-art in distributed least-squares linear regression in cloud networks.

Secure/private *multiparty computing* (MPC) and machine learning (e.g., [26, 27]) are also extensively studied topics that address a problem setting similar to LCC. As we elaborate in Section 3, com-

pared with conventional methods (e.g., the celebrated BGW scheme for MPC [26]), LCC achieves substantial reduction in the amount of randomness, storage overhead, and computation complexity.

2 Problem Formulation and Examples

We consider the problem of evaluating a multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a dataset $X = (X_1, \ldots, X_K)$, where $\mathbb{V}$ and $\mathbb{U}$ are vector spaces over a field $\mathbb{F}$. The goal is to compute $Y_1 \triangleq f(X_1), \ldots, Y_K \triangleq f(X_K)$, in a distributed computing environment with a master and N workers (Figure 1). We define the degree of f, denoted by $\deg f$, as the *total degree* of the polynomial.[1]

In this setting each worker has already stored a fraction of the dataset prior to computation, in a possibly coded manner. Specifically, for $i \in [N]$ (where $[N] \triangleq \{1, \ldots, N\}$), worker i stores $\tilde{X}_i \triangleq g_i(X_1, \ldots, X_K)$, where g_i is a (possibly random) function, refered to as the encoding function of that worker. We restrict our attention to linear encoding schemes (see [28] for a formal definition) , which guarantee low encoding complexity and simple implementation.

Each worker i computes $\tilde{Y}_i \triangleq f(\tilde{X}_i)$ and returns it to the master. The master waits for a subset of fastest workers and decodes $Y_1, \ldots, Y_K$. The procedure must satisfy several additional requirements:

- **Resiliency**, i.e., robustness against stragglers. Formally, the master must be able to obtain the correct values of $Y_1, \ldots, Y_K$ even if up to S workers fail to respond (or respond after the master executes the decoding algorithm), where S is the *resiliency parameter*. A scheme that guarantees resiliency against S stragglers is called S*-resilient*.
- **Security**, i.e., robustness against adversaries. That is, the master must be able to obtain correct values of $Y_1, \ldots, Y_K$ even if up to A workers return arbitrarily erroneous results, where A is the *security parameter*. A scheme that guarantees security against A adversaries is called A*-secure*.
- **Privacy**, i.e., the workers must remain oblivious to the content of the dataset, even if up to T of them collude, where T is the *privacy parameter*. Formally, for every $\mathcal{T} \subseteq [N]$ of size at most T, we must have $I(X; \tilde{X}_{\mathcal{T}}) = 0$, where I is mutual information, $\tilde{X}_{\mathcal{T}}$ represents the collection of the encoded dataset stored at the workers in $\mathcal{T}$, and X is seen as chosen uniformly at random. A scheme which guarantees privacy against T colluding workers is called T*-private*. [2]

More concretely, given any subset of workers that return the computing results (denoted by $\mathcal{K}$), the master computes $(\hat{Y}_1, ..., \hat{Y}_K) = h_{\mathcal{K}}(\{\tilde{Y}_i\}_{i \in \mathcal{K}})$, where each $h_{\mathcal{K}}$ is a deterministic function (or is random but independent of both the encoding functions and input data). We refer to the $h_{\mathcal{K}}$'s as *decoding functions*.[3] We say that a scheme is S-resilient, A-secure, and T-private if the master always returns the correct results (i.e., each $Y_i = \hat{Y}_i$), and all above requirements are satisfied.

Given this framework, we aim to characterize the region for (S, A, T), such that an S-resilient, A-secure, and T-private scheme can be found, given N, K, and function f, for any sufficiently large field $\mathbb{F}$. This framework encapsulates many computations of interest, including linear computation [9, 12, 29, 30], bilinear computation [31], general tensor algebra [32], and gradient computation [33].

3 Main Results and Prior Works

We now state our main results and discuss their connections with prior works. Our first theorem characterizes the region for (S, A, T) that LCC achieves (i.e., the set of all feasible S-resilient, A-secure, and T-private schemes via LCC as defined in the previos section).

Theorem 1. *Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, Lagrange Coded Computing (LCC) provides an S-resilient, A-secure, and T-private scheme for computing $\{f(X_i)\}_{i=1}^K$ for any polynomial f, as long as*

$$(K + T - 1) \deg f + S + 2A + 1 \leq N. \tag{1}$$

[1] We focus on the non-trivial case where $K > 0$ and f is not constant. The *total degree* of a polynomial f is the maximum among all the total degrees of its monomials. When discussing finite $\mathbb{F}$, we resort to the canonical representation of polynomials, in which the individual degree within each term is no more than $(|\mathbb{F}| - 1)$.

[2] Equivalently, it requires that $\tilde{X}_{\mathcal{T}}$ and X are independent. Under this condition, the input data X still appears uniformly random after the colluding workers learn $\tilde{X}_{\mathcal{T}}$, which guarantees the privacy. To guarantee that the privacy requirement is well defined, we assume that $\mathbb{F}$ and $\mathbb{V}$ are finite whenever $T > 0$.

[3] Similar to encoding, we also require the decoding function to have low complexity. When there is no adversary ($A = 0$), we restrict our attention to *linear decoding schemes*.

Remark 1. To prove Theorem 1, we formally present LCC in Section 4, which achieves the stated resiliency, security, and privacy. The key idea is to encode the input dataset using the Lagrange polynomial. In particular, encoding functions (i.e., g_i's) in LCC amount to evaluations of a Lagrange polynomial of degree $K-1$ at N distinct points. Hence, computations at the workers amount to evaluations of a *composition* of that polynomial with the desired function f. Therefore, inequality (1) may simply be seen as the number of evaluations that are necessary and sufficient in order to interpolate the composed polynomial, which is later evaluated at a certain point to finalize the computation. LCC also has a number of additional properties of interest. First, the proposed encoding is *identical* for all computations f, which allows pre-encoding of the data without knowing the identity of the computing task (i.e., universality). Second, decoding and encoding rely on polynomial interpolation and evaluation, and hence efficient off-the-shelf subroutines can be used.

Remark 2. Note that LHS of inequality (1) is independent of the number of workers N, hence the key property of LCC is that adding 1 worker can increase its resilience to stragglers by 1 or its security to malicious servers by $1/2$, while keeping the privacy constraint T the same. Note that using an uncoded replication based approach, to increase the resiliency to stragglers by 1, one needs to essentially repeat *each* computation once more (i.e., requiring K more machines as opposed to 1 machine in LCC). This result essentially extends the well-known optimal scaling of error-correcting codes (i.e., adding one parity can provide robustness against one erasure or $1/2$ error in optimal maximum distance separable codes) to the distributed computing paradigm.

Our next theorem demonstrates the optimality of LCC. [4]

Theorem 2. *Lagrange Coded Computing (LCC) achieves the optimal trade-off between resiliency, security, and privacy for any multilinear function f among all computing schemes that uses linear encoding, for all problem scenarios. Moreover, when focusing on the case where no security constraint is imposed, LCC, which uses linear decoding functions, is optimal for any polynomial f among all schemes with this additional decoding constrain, when the characteristic of $\mathbb{F}$ is sufficiently large.*

Remark 3. Theorem 2 is formally stated and proved in the long version [28]. The main proof idea is to show that any computing strategy that outperforms LCC (or its uncoded version) would violate the decodability requirement, by finding two instances of the computation process where the same intermediate computing results correspond to different output values.

Remark 4. LCC improves and generalizes previously works on coded computing in a few aspects: *Generality*–LCC significantly generalizes prior works to go beyond linear and bilinear computations that have so far been the main focus in this area, and can be applied to arbitrary multivariate polynomial computations that arise in machine learning applications. *Universality*–once the data has been coded, any polynomial up to a certain degree can be computed distributedly via LCC. In other words, data encoding of LCC can be *universally* used for any polynomial computation. This is in stark contrast to previous task specific coding techniques in the literature. Furthermore, workers apply the same computation as if no coding took place; a feature that reduces computational costs, and prevents ordinary servers from carrying the burden of outliers. *Security and Privacy*–other than a handful of works discussed above, straggler mitigation (i.e., resiliency) has been the primary focus of the coded computing literature. This work extends the application of coded computing to secure and private computing for general polynomial computations.

Remark 5. To illustrate the significant role of LCC in secure and private computing, we consider the celebrated BGW scheme [26]. [5] As we elaborate below, in comparison with the BGW scheme, LCC results in a factor of K reduction in the amount of randomness, storage overhead, and computation complexity, while requiring more workers to guarantee the same level of privacy (see Table 1). [6]

A key distinction between the two is that BGW uses Shamir's scheme [34] to secret-share the *entire* dataset, while LCC instead uses Lagrange polynomials for encoding. LCC operates on $\frac{1}{K}$ fraction of the input dataset as a unit, resulting in a significant reduction (factor of K) in storage and randomness.

[4] Here LCC refers to the better between the achievability scheme for Theorem 1 and its uncoded version, for details, see [28].

[5] Conventionally, the BGW scheme operates in multiple rounds, requiring significantly more communication overhead than one-shot schemes. For simplicity of comparison, we present a modified one-shot version of BGW.

[6] A BGW scheme was also proposed in [26] for secure MPC, however for a substantially different setting. Similarly, a comparison can be made by adapting it to our setting, leading to similar results, omitted for brevity.

The BGW scheme will then evaluate f over *all* stored coded data at the nodes. In LCC, however, each worker ℓ only needs to store *one* encoded $\tilde{X}_\ell$ and compute $f(\tilde{X}_\ell)$. This leads to the second key advantage of LCC, which is a factor of K reduction in computation complexity at each worker.

	BGW	LCC
Complexity per worker	K	1
Frac. data per worker	1	$1/K$
Randomness	KT	T
Min. num. of workers	$\deg(f)(T+1)$	$\deg(f)(K+T-1)+1$

Table 1: Comparison between BGW based designs and LCC. The computational complexity is normalized by that of evaluating f; randomness, which refers to the number of random entries used in encoding functions, is normalized by the length of X_i.

In BGW, after computation each worker ℓ has essentially evaluated a polynomial of degree at most $\deg(f) \cdot T$. With no straggler or adversary, the master can recover all required results through polynomial interpolation, as long as $N \geq \deg(f) \cdot T + 1$ workers participated[7]. Under the same condition, LCC requires $\deg(f) \cdot (K+T-1)+1$ workers, larger than that of the BGW scheme.

4 Lagrange Coded Computing

In this Section we prove Theorem 1 by presenting LCC and characterizing the region for (S, A, T) that it achieves. We start with an example to illustrate the key components of LCC.

4.1 Illustrating Example

Consider the function $f(X_i) = X_i^2$, where X_i's are some square matrices. We demonstrate LCC in the scenario where the input data X is partitioned into $K = 2$ batches X_1 and X_2, and the computing system has $N = 8$ workers. Assume that $\mathbb{F} = \mathbb{F}_{11}$. We aim to achieve $(S, A, T) = (1, 1, 1)$.

The gist of LCC is picking a uniformly random matrix Z, and encoding (X_1, X_2, Z) using a Lagrange interpolation polynomial: $u(z) \triangleq X_1 \cdot \frac{(z-2)(z-3)}{(1-2)(1-3)} + X_2 \cdot \frac{(z-1)(z-3)}{(2-1)(2-3)} + Z \cdot \frac{(z-1)(z-2)}{(3-1)(3-2)}$. We then fix distinct $\{\alpha_i\}_{i=1}^8$ in $\mathbb{F}$ such that $\{\alpha_i\}_{i=1}^8 \cap [2] = \varnothing$, and let workers $1, \ldots, 8$ store $u(\alpha_1), \ldots, u(\alpha_8)$.

First, note that for every $j \in [8]$, worker j sees $\tilde{X}_j$, a linear combination of X_1 and X_2 that is masked by addition of $\lambda \cdot Z$ for some nonzero $\lambda \in \mathbb{F}_{11}$; since Z is uniformly random, this guarantees perfect privacy for $T = 1$. Next, note that worker j computes $f(\tilde{X}_j) = f(u(\alpha_j))$, which is an evaluation of the composition polynomial $f(u(z))$, whose degree is at most 4, at α_j.

Normally, a polynomial of degree 4 can be interpolated from 5 evaluations at distinct points. However, the presence of $A = 1$ adversary and $S = 1$ straggler requires the master to employ a Reed-Solomon decoder, and have *three* additional evaluations at distinct points (in general, two additional evaluations per adversary and one per straggler). Finally, after decoding polynomial $f(u(z))$, the master can obtain $f(X_1)$ and $f(X_2)$ by evaluating it at $z = 1$ and $z = 2$.

4.2 General Description

For brevity, we present the main idea of the general LCC construction, while a formal version is provided in the long version [28]. Similar to Subsection 4.1, we select any $K + T$ distinct elements $\beta_1, \ldots, \beta_{K+T}$ from $\mathbb{F}$, and find a polynomial $u : \mathbb{F} \to \mathbb{V}$ of degree at most $K + T - 1$ such that $u(\beta_i) = X_i$ for any $i \in [K]$, and $u(\beta_i) = Z_i$ for $i \in \{K+1, \ldots, K+T\}$, where all Z_i's are chosen uniformly at random from $\mathbb{V}$. We then select N distinct elements $\{\alpha_i\}_{i\in[N]}$ from $\mathbb{F}$ such that $\{\alpha_i\}_{i\in[N]} \cap \{\beta_j\}_{j\in[K]} = \varnothing$, and let $\tilde{X}_i = u(\alpha_i)$ for any $i \in [N]$. After each worker i applies f on $\tilde{X}_i$, it returns an evaluation of a polynomial of degree $\deg(f) \cdot (K+T-1)$. As we prove in [28], as long as sufficiently many workers return the computation results (as specified in Theorem 1), the master can successfully recover the needed results using Reed-Solomon decoding, in spite of S stragglers and A adversaries. This construction also guarantees T-privacy.

5 Application to Linear Regression and Experiments on AWS EC2

We demonstrate a practical application of LCC in accelerating distributed linear regression, whose gradient computation is a quadratic function of the input, hence matching well the LCC framework. We also show its performance gain over state of the arts via experiments on AWS EC2 clusters.

[7] It is also possible to use the conventional multi-round BGW, which only requires $N \geq 2T + 1$ workers to ensure T-privacy. However, multiple rounds of computation and communication ($\Omega(\log \deg(f))$ rounds) are needed, which further increases its communication overhead.

Applying LCC for linear regression. Given a feature matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ containing m data points of d features, and a label vector $\boldsymbol{y} \in \mathbb{R}^m$, a linear regression problem aims to find the weight vector $\boldsymbol{w} \in \mathbb{R}^d$ that minimizes the loss $||\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}||^2$. Gradient descent (GD) solves this by iteratively moving the weight along the negative gradient, which is in iteration-t computed as $2\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}^{(t)}-\boldsymbol{y})$.

To run GD distributedly over a system with a master and n workers, each worker stores $\frac{r}{n}$ fraction of coded columns for some parameter $1 \le r \le n$. Given the current weight $\boldsymbol{w}$, each worker performs computation using its local storage. We cast this computation to the model in Section 2, by grouping the columns into $K{=}\lceil\frac{n}{r}\rceil$ blocks such that $\boldsymbol{X} = [\bar{\boldsymbol{X}}_1 \cdots \bar{\boldsymbol{X}}_K]^\top$, and let the system compute $\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{w}$, which is the sum of a degree-2 polynomial $f(\bar{\boldsymbol{X}}_k) = \bar{\boldsymbol{X}}_k\bar{\boldsymbol{X}}_k^\top\boldsymbol{w}$ evaluated over $\bar{\boldsymbol{X}}_1, \ldots, \bar{\boldsymbol{X}}_K$.[8] Using LCC, we can achieve a recovery threshold of $R_{\text{LCC}} = 2(K-1)+1 = 2\lceil\frac{n}{r}\rceil - 1$ (Theorem 1). [9]

Comparison with state of the arts. The conventional uncoded scheme picks $r = 1$, and has each worker j compute $\bar{\boldsymbol{X}}_j\bar{\boldsymbol{X}}_j^\top\boldsymbol{w}$, yielding recovery threshold $R_{\text{uncoded}} = n$. By redundantly storing/processing $r > 1$ *uncoded* sub-matrices at each worker, the "gradient coding" (GC) methods [10, 14, 15] code across partial gradients computed from uncoded data, and reduce the recovery threshold to $R_{\text{GC}} = n - r + 1$. An alternative "matrix-vector multiplication based" (MVM) approach [17] requires two rounds of computation. MVM achieves a recovery threshold of $R_{\text{MVM}} = \lceil\frac{2n}{r}\rceil$ in *each* round, when the storage is evenly split between rounds.

Compared with GC, LCC codes directly on data, and reduces the recovery threshold by about $r/2$ times. While the amount of computation and communication per node is the same for both, LCC is expected to finish much faster due to its much smaller recovery threshold. Compared with MVM, LCC achieves a smaller recovery threshold than that in each round of MVM (assuming even storage split). While each MVM worker performs less computation in each iteration, it sends two vectors with sizes proportional to m and d respectively, whereas each LCC worker only sends one dimension-d vector.

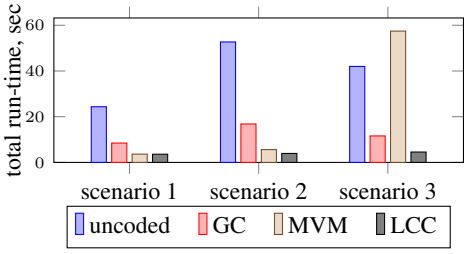

Figure 2: Run-time comparison of LCC with other three schemes: uncoded, GC, and MVM.

We run linear regression on EC2 using Nesterov's accelerated gradient descent, implemented on `t2.micro` nodes. We generate synthetic datasets of m data points, by randomly sampling 1) true weight $\boldsymbol{w}^*$, and 2) each input $\boldsymbol{x}_i$ of d features and computing output $y_i = \boldsymbol{x}_i^\top\boldsymbol{w}^*$. For each dataset, we run GD for 100 iterations over 40 workers. We use different dimensions of input $\boldsymbol{X}$: 8000×7000 for scenarios 1&2, and 160000×500 for scenario 3. In scenario 1, we let the system run with natural stragglers. To mimic slow/failed workers, we artificially introduce stragglers in scenarios 2 and 3, by imposing a 0.5 seconds delay on each worker with probability 5% in each iteration. To avoid numerical instability due to large entries, we can embed input data into a large finite field, and apply LCC with exact computations. However in all our experiments the gradients are calculated correctly without carrying out this step.

Results. For GC and LCC, we optimize the total run-time over r subject to local memory size. For MVM, we further optimize run-time over the storage assigned between two rounds of multiplications. See Fig. 2 for run-time measurement, and the long version [28] for detailed breakdown by scenario.

We draw the following conclusions from the experiments.

- LCC achieves the least run-time in all scenarios. In particular, LCC speeds up the uncoded scheme by 6.79×-13.43×, the GC scheme by 2.36-4.29×, and the MVM scheme by 1.01-12.65×.
- In scenarios 1 & 2 where m is close to the number of features d, LCC achieves a similar performance as MVM. However, with much more data points in scenario 3, LCC finishes substantially faster than MVM by as much as 12.65×. The main reason is that MVM requires large amounts of data transfer from workers to the master in the first round and from master to workers in the second round (both proportional to m). However, the amount of communication from each worker or master is proportional to d for all other schemes, much smaller than m in scenario 3.

[8] Since the value of $\boldsymbol{X}^\top\boldsymbol{y}$ does not vary across iterations, it only needs to be computed once. We assume that it is available at the master for weight updates.

[9] The *recovery threshold* (denoted by R) is defined as the minimum number of workers the master needs to wait for, to guarantee decodability (i.e., tolerating the remaining stragglers).

Acknowledgement This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, and NSF grants CCF-1703575 and CCF-1763673. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Qian Yu is supported by the Google PhD Fellowship.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[2] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[3] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, (Cambridge, MA, USA), pp. 19–27, MIT Press, 2014.

[4] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *Journal of Machine Learning Research*, vol. 17, no. 106, pp. 1–37, 2016.

[5] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, pp. 19–27, 2014.

[6] P. Blanchard, R. Guerraoui, J. Stainer, *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, pp. 118–128, 2017.

[7] R. Cramer, I. B. Damgrd, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. New York, NY, USA: Cambridge University Press, 1st ed., 2015.

[8] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, (Berlin, Heidelberg), pp. 192–206, Springer-Verlag, 2008.

[9] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, pp. 1514–1529, March 2018.

[10] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3368–3376, 2017.

[11] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with ldpc codes," *SysML Conference*, 2018.

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5440–5448, 2017.

[13] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *arXiv preprint arXiv:1710.09990*, 2017.

[14] W. Halbawi, N. A. Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *CoRR*, vol. abs/1706.05436, 2017.

[15] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," *arXiv preprint arXiv:1707.03858*, 2017.

[16] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *arXiv preprint arXiv:1802.03475*, 2018.

[17] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *NIPS Workshop on Machine Learning Systems*, Dec. 2015.

[18] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proceedings of the 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 964–971, Sept. 2015.

[19] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, pp. 2092–2100, 2016.

[20] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, pp. 4406–4416, Curran Associates, Inc., 2017.

[21] Q. Yu, S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "How to optimally allocate resources for coded distributed computing?," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2017.

[22] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2018.

[23] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. R. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[24] H. A. Nodehi and M. A. Maddah-Ali, "Limited-sharing multi-party computation for massive matrix operations," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1231–1235, June 2018.

[25] L. Chen, Z. Charles, D. Papailiopoulos, *et al.*, "Draco: Robust distributed training via redundant gradients," *arXiv preprint arXiv:1803.09877*, 2018.

[26] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 1–10, ACM, 1988.

[27] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 19–38, May 2017.

[28] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint arXiv:1806.00939*, 2018.

[29] R. Bitar, P. Parag, and S. E. Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *arXiv preprint arXiv:1802.02640*, 2018.

[30] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," *arXiv preprint arXiv:1804.09791*, 2018.

[31] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint arXiv:1801.07487*, 2018.

[32] P. Renteln, *Manifolds, Tensors, and Forms: An Introduction for Mathematicians and Physicists*. Cambridge University Press, 2013.

[33] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[34] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.