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

Qian Yu[*], **Songze Li**[*], **Netanel Raviv**[†], **Seyed Mohammadreza 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, i.e., in terms of tolerating the maximum number of stragglers and adversaries, and providing data privacy against the maximum number of colluding workers. 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×, and also achieves a 2.36×-12.65× speedup over the state-of-the-art straggler mitigation strategies.

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89. Copyright 2019 by the author(s).

1 Introduction

The massive size of modern datasets necessitates computational tasks to be performed in a distributed fashion, where the data is dispersed among many servers that operate in parallel [Abadi et al., 2016]. As we "scale out" computations across many servers, however, several fundamental challenges arise. Cheap commodity hardware tends to vary greatly in computation time, and it has been demonstrated [Dean and Barroso, 2013, Li et al., 2014a, Yadwadkar et al., 2016] that a small fraction of servers, referred to as *stragglers*, can be 5 to 8 times slower than the average, thus 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 [Li et al., 2014b]. Distributed computing systems are also much more susceptible to adversarial servers, making security and privacy a major concern [Blanchard et al., 2017, Cramer et al., 2015, Bogdanov et al., 2008].

We consider a general scenario in which the 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.

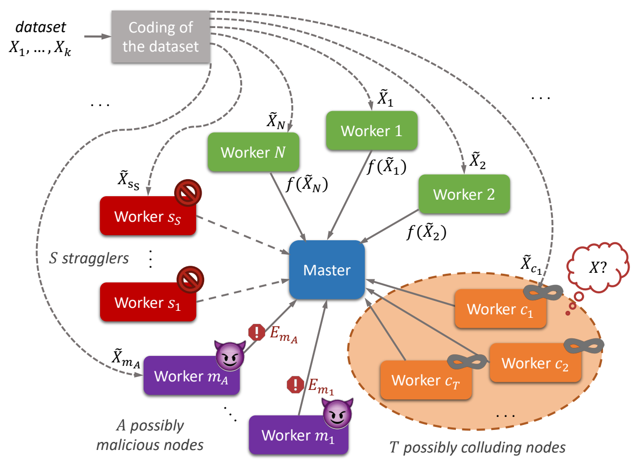

Figure 1: An overview of the problem considered in this paper, where the goal is to evaluate a *not necessarily linear* function f on a given dataset $X = (X_1, X_2, \dots, X_K)$ using N workers. Each worker applies f on a possibly *coded* version of the inputs (denoted by $\tilde{X}_i$'s). By carefully 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 $c_1, ..., c_T$).

LCC can be applied to any computation scenario in which the function of interest is an *arbitrary multivariate polynomial* of the input dataset. This covers many computations of interest in machine learning, such as various gradient and loss-function computations in learning algorithms and tensor algebraic operations (e.g., low-rank tensor approximation). The key idea of LCC is to encode the input dataset using the well-known Lagrange polynomial, in order to create computational redundancy in a novel coded form across the workers. This redundancy can then be exploited to provide resiliency to stragglers, security against malicious servers, and privacy of the dataset.

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 $X = (X_1, X_2, \dots, X_K)$, where f is a given multivariate polynomial with degree $\deg f$. To do so, N coded versions of the input dataset, denoted by $\tilde{X}_1, \tilde{X}_2, \dots, \tilde{X}_N$ 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.

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. In other words, any computing scheme (under certain complexity constrains on the encoding and decoding designs) can achieve (S, A, T) if and only if $(K + T - 1)\deg f + S + 2A + 1 \leq N$.[1] This result further extends the scaling law in coding theory to private computing, showing that any additional worker enables data privacy against $1/\deg f$ additional colluding workers.

Finally, we specialize our general theoretical guarantees for LCC in the context of least-squares linear regression, which is one of the elemental learning tasks, and demonstrate its performance gain by optimally suppressing stragglers. Leveraging the algebraic structure of gradient computations, several strategies have been developed recently to exploit data and gradient coding for straggler mitigation in the training process (see, e.g., [Lee et al., 2018, Tandon et al., 2017, Maity et al., 2018, Karakus et al., 2017, Li et al., 2017a]). We implement LCC for regression on Amazon EC2 clusters, and empirically compare its performance with the conventional uncoded approaches, and two state-of-the-art straggler mitigation schemes: gradient coding (GC) [Tandon et al., 2017, Halbawi et al., 2017, Raviv et al., 2017, Ye and Abbe, 2018] and matrix-vector multiplication (MVM) based approaches [Lee et al., 2018, Maity et al., 2018]. Our experiment results demonstrate that compared with the uncoded scheme, LCC improves the run-time by 6.79×-13.43×. Compared with the GC scheme, LCC improves the run-time by 2.36×-4.29×. Compared with the MVM scheme, LCC improves the run-time by 1.01×-12.65×.

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., [Lee et al., 2015, Li et al., 2015, Yu et al., 2017a, Li et al., 2018a, Dutta et al., 2016, Yu et al., 2017b, Tandon et al., 2017, Halbawi et al., 2017, Raviv

[1]More accurately, when $N < K\deg f - 1$, we prove that the optimal tradeoff is instead given by $K(S + 2A + \deg f \cdot T + 1) \leq N$, which can be achieved by a variation of the LCC scheme, as described in Appendix D.

et al., 2017, Dutta et al., 2018, Nodehi and Maddah-Ali, 2018, Chen et al., 2018]). As we discuss in more detail in Section 3.1, 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×-12.65× speedup over the state-of-the-art in distributed least-squares linear regression in cloud networks.

Secure *multiparty computing* (MPC) and secure/private Machine Learning (e.g., [Ben-Or et al., 1988, Mohassel and Zhang, 2017]) are also extensively studied topics that address a problem setting similar to LCC. As we elaborate in Section 3.1, compared with conventional methods in this area (e.g., the celebrated BGW scheme for secure/private MPC [Ben-Or et al., 1988]), LCC achieves substantial reduction in the amount of randomness, storage overhead, and computation complexity.

2 Problem Formulation

We consider the problem of evaluating a multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a dataset $X = (X_1, \ldots, X_K)$,[2] where $\mathbb{V}$ and $\mathbb{U}$ are vector spaces of dimensions M and L, respectively, over the field $\mathbb{F}$. We assume a distributed computing environment with a master and N workers (Figure 1), in which the goal is to compute $Y_1 \triangleq f(X_1), \ldots, Y_K \triangleq f(X_K)$. We denote the *total degree*[3] of the polynomial f by $\deg f$.

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[4], which guarantee low encoding complexity and simple implementation.

Each worker $i \in [N]$ computes $\tilde{Y}_i \triangleq f(\tilde{X}_i)$ and returns the result to the master. The master waits for a subset of fastest workers and then decodes $Y_1, \ldots, Y_K$. This 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* of the system. 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* of the system. 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* of the system. 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.[5] A scheme which guarantees privacy against T colluding workers is called T*-private.* [6]

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.*[7] 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 the above 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 parameters N, K, and function f, for any sufficiently large field $\mathbb{F}$.

This framework encapsulates many computation tasks of interest, including linear computation [Dutta et al., 2016, Bitar et al., 2018, Karakus et al., 2017, Lee et al., 2018, Wang et al., 2018], bilinear computation [Yu et al., 2018], general tensor algebra [Renteln, 2013], and gradient computation [Shalev-Shwartz and Ben-David, 2014], and many are not studied by state-of-the-art coded computing frameworks.

[2] We focus on the non-trivial case where $K > 0$ and f is not constant.

[3] 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 $(|\mathbb{F}| - 1)$.

[4] A formal definition is provided in Section 5.

[5] 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.

[6] To guarantee that the privacy requirement is well defined, we assume that $\mathbb{F}$ and $\mathbb{V}$ are finite whenever $T > 0$.

[7] 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.*

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 $X = (X_1, \ldots, X_K)$, 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*

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

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 well-known 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.[8]

Remark 2. Besides the approach presented to achieve Theorem 1, a variation of LCC can be used to achieve any (S, A, T) as long as $K(S + 2A + \deg f \cdot T + 1) \leq N$. This scheme (presented in Appendix D) achieves an improved region when $N < K\deg f - 1$ and $T = 0$, where it recovers the *uncoded repetition* scheme. For brevity, we refer the better of these two scheme as LCC when presenting optimality results (i.e., Theorem 2).

Remark 3. 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.

[8]A more detailed discussion on the coding complexities of LCC can be found in Appendix B.

Our next theorem demonstrates the optimality of LCC.

Theorem 2. *LCC achieves the optimal trade-off between resiliency, security, and privacy (i.e., achieving the largest region of (S,A,T)) 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 is optimal for any polynomial f among all schemes with additional constraints of linear decoding and sufficiently large (or zero) characteristic of $\mathbb{F}$.*

Remark 4. Theorem 2 is proved in Section 5. The main proof idea is to show that any computing strategy that outperforms LCC 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 5. In addition to the result we show in Theorem 2, we can also prove that LCC achieves optimality in terms of the amount of randomness used in data encoding. Specifically, we show in Appendix I that LCC requires injecting the minimum amount of randomness, among all computing schemes that universally achieve the same resiliency-security-privacy tradeoff for all linear functions f.

We conclude this section by discussing several lines of related work in the literature and contrasting them with LCC.

3.1 LCC vs. Prior Works

The study of coding theoretic techniques for accelerating large scale distributed tasks (a.k.a. *coded computing*) was initiated in [Lee et al., 2015, Li et al., 2015, Li et al., 2018a]. Following works focused largely on matrix-vector and matrix-matrix multiplication (e.g., [Dutta et al., 2016, Yu et al., 2017b, Dutta et al., 2018, Yu et al., 2018]), gradient computation in gradient descent algorithms (e.g., [Tandon et al., 2017, Raviv et al., 2017, Li et al., 2017a]), communication reduction via coding (e.g., [Li et al., 2017b, Ezzeldin et al., 2017, Prakash et al., 2018, Konstantinidis and Ramamoorthy, 2018]), and secure and private computing (e.g., [Nodehi and Maddah-Ali, 2018, Chen et al., 2018]).

LCC recovers several previously studied results as special cases. For example, setting f to be the identity function and $\mathbb{V} = \mathbb{U}$ reduces to the well-studied case of *distributed storage*, in which Theorem 1 is well known (e.g., the Singleton bound [Roth, 2006, Thm. 4.1]). Further, f can correspond to matrix-vector and matrix-

matrix multiplication, in which the special cases of Theorem 1 are known as well [Lee et al., 2018, Yu et al., 2018].

More importantly, LCC improves and generalizes these 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. In fact, many specific computations considered in the past can be seen as special cases of polynomial computation. This includes matrix-vector multiplication, matrix-matrix multiplication, and gradient computation whenever the loss function at hand is a polynomial, or is approximated by one. *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.

Providing security and privacy for *multiparty computing* (MPC) and Machine Learning systems is an extensively studied topic which addresses a problem setting similar to LCC. To illustrate the significant role of LCC in secure and private computing, let us consider the celebrated BGW MPC scheme [Ben-Or et al., 1988]. [9]

Given inputs $\{X_i\}_{i=1}^K$, BGW first uses Shamir's scheme [Shamir, 1979] to encode the dataset in a privacy-preserving manner as $P_i(z) = X_i + Z_{i,1}z + \ldots + Z_{i,T}z^T$ for every $i \in [K]$, where $Z_{i,j}$'s are i.i.d uniformly random variables and T is the number of colluding workers that should be tolerated. The key distinction between the data encoding of BGW scheme and LCC is that we instead use Lagrange polynomials to encode the data. This results in significant reduction in the amount of randomness needed in data encoding (BGW needs KT $z_{i,j}$'s while as we describe in the next section, LCC only needs T amount of randomness).

The BGW scheme will then store $\{P_i(\alpha_\ell)\}_{i\in[K]}$ to worker ℓ for every $\ell \in [N]$, given some distinct values $\alpha_1, \ldots, \alpha_N$. The computation is then carried out by evaluating f over *all* stored coded data at the nodes. In the LCC scheme, on the other hand, each worker ℓ only needs to store *one* encoded data ($\tilde{X}_\ell$) and compute $f(\tilde{X}_\ell)$. This gives rise to the second key advantage of LCC, which is a factor of K in storage overhead and 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.

After computation, each worker ℓ in the BGW scheme has essentially evaluated the polynomials $\{f(P_i(z))\}_{i=1}^K$ at $z = \alpha_\ell$, whose degree is at most $\deg(f) \cdot T$. Hence, if no straggler or adversary appears (i.e, $S = A = 0$), the master can recover all required results $f(P_i(0))$'s, through polynomial interpolation, as long as $N \geq \deg(f) \cdot T + 1$ workers participated in the computation[10]. Note that under the same condition, LCC scheme requires $N \geq \deg(f) \cdot (K + T - 1) + 1$ number of workers, which is larger than that of the BGW scheme.

Hence, in overall 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. This is summarized in Table 1.[11]

Recently, [Nodehi and Maddah-Ali, 2018] has also combined ideas from the BGW scheme and [Yu et al., 2017b] to form *polynomial sharing*, a private coded computation scheme for arbitrary matrix polynomials. However, polynomial sharing inherits the undesired BGW property of performing a communication round for *every* bilinear operation in the polynomial; a fea-

[9]Conventionally, the BGW scheme operates in a multi-round fashion, requiring significantly more communication overhead than one-shot approaches. For simplicity of comparison, we present a modified one-shot version of BGW.

[10]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.

[11]A BGW scheme was also proposed in [Ben-Or et al., 1988] 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, which we omit for brevity.

ture that drastically increases communication overhead, and is circumvented by the one-shot approach of LCC. *DRACO* [Chen et al., 2018] is also recently proposed as a secure computation scheme for gradients. Yet, DRACO employs a blackbox approach, i.e., the resulting gradients are encoded rather than the data itself, and the inherent algebraic structure of the gradients is ignored. For this approach, [Chen et al., 2018] shows that a $2A+1$ *multiplicative* factor of redundant computations is necessary. In LCC however, the blackbox approach is disregarded in favor of an algebraic one, and consequently, a $2A$ *additive* factor suffices.

LCC has also been recently applied to several applications in which security and privacy in computations are critical. For example, in [Li et al., 2018b], LCC has been applied to enable a scalable and secure approach to sharding in blockchain systems. Also, in [So et al., 2019], a privacy-preserving approach for machine learning has been developed that leverages LCC to provides substantial speedups over cyrptographic approaches that relay on MPC.

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.[12] 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 input X_i's are $\sqrt{M}\times\sqrt{M}$ square matrices for some square integer M. 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. In addition, the suggested scheme is 1-resilient, 1-secure, and 1-private (i.e., achieves $(S,A,T)=(1,1,1)$).

The gist of LCC is picking a uniformly random matrix Z, and encoding (X_1,X_2,Z) using a Lagrange interpolation polynomial:[13]

$$u(z)\triangleq X_1\cdot\frac{(z-2)(z-3)}{(1-2)(1-3)}+X_2\cdot\frac{(z-1)(z-3)}{(2-1)(2-3)}+ Z\cdot\frac{(z-1)(z-2)}{(3-1)(3-2)}.$$

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 for every adversary and one for every 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

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 $\mathbb{V}$. This is simply accomplished by letting u be the *Lagrange interpolation polynomial*

$$u(z)\triangleq\sum_{j\in[K]}X_j\cdot\prod_{k\in[K+T]\setminus\{j\}}\frac{z-\beta_k}{\beta_j-\beta_k}+ \sum_{j=K+1}^{K+T}Z_j\cdot\prod_{k\in[K+T]\setminus\{j\}}\frac{z-\beta_k}{\beta_j-\beta_k}.$$

We then select N distinct elements $\{\alpha_i\}_{i\in[N]}$ from $\mathbb{F}$ such that $\{\alpha_i\}_{i\in[N]}\cap\{\beta_j\}_{j\in[K]}=\varnothing$ (this requirement is alleviated if $T=0$), and let $\tilde{X}_i=u(\alpha_i)$ for any $i\in[N]$. That is, the input variables are encoded as

$$\tilde{X}_i=u(\alpha_i)=(X_1,\ldots,X_K,Z_{K+1},\ldots,Z_{K+T})\cdot U_i, \quad (2)$$

where $U\in\mathbb{F}_q^{(K+T)\times N}$ is the encoding matrix $U_{i,j}\triangleq\prod_{\ell\in[K+T]\setminus\{i\}}\frac{\alpha_j-\beta_\ell}{\beta_i-\beta_\ell}$, and U_i is its i'th column.[14]

Following the above encoding, each worker i applies f on $\tilde{X}_i$ and sends the result back to the master. Hence, the master obtains $N-S$ evaluations, at most A of which are incorrect, of the polynomial $f(u(z))$. Since $\deg(f(u(z)))\leq\deg(f)\cdot(K+T-1)$, and $N\geq(K+T-1)\deg(f)+S+2A+1$, the master can obtain all coefficients of $f(u(z))$ by applying Reed-Solomon decoding. Having this polynomial, the master evaluates it at β_i for every $i\in[K]$ to obtain $f(u(\beta_i))=f(X_i)$, and hence we have shown that the above scheme is S-resilient and A-secure.

As for the T-privacy guarantee of the above scheme, our proof relies on the fact that the bottom $T\times N$

[12] For an algorithmic illustration, see Appendix A.

[13] Assume that $\mathbb{F}$ is a finite field with 11 elements.

[14] By selecting the values of α_i's differently, we can recover the uncoded repetition scheme, see Appendix D.

submatrix U^{bottom} of U is an MDS matrix (i.e., every $T \times T$ submatrix of U^{bottom} is invertible, see Lemma 2 in the supplementary material). Hence, for a colluding set of workers $\mathcal{T} \subseteq [N]$ of size T, their encoded data $\tilde{X}_{\mathcal{T}}$ satisfies $\tilde{X}_{\mathcal{T}} = XU^{top}_{\mathcal{T}} + ZU^{bottom}_{\mathcal{T}}$, where $Z \triangleq (Z_{K+1}, \ldots, Z_{K+T})$, and $U^{top}_{\mathcal{T}} \in \mathbb{F}_q^{K\times T}$, $U^{bottom}_{\mathcal{T}} \in \mathbb{F}_q^{T\times T}$ are the top and bottom submatrices which correspond to the columns in U that are indexed by $\mathcal{T}$. Now, the fact that any $U^{bottom}_{\mathcal{T}}$ is invertible implies that the random padding added for these colluding workers is uniformly random, which completely masks the coded data $XU^{top}_{\mathcal{T}}$. This directly guarantees T-privacy.

5 Optimality of LCC

In this section, we provide a layout for the proof of optimality for LCC (i.e., Theorem 2). Formally, we define that a *linear encoding function* is one that computes a linear combination of the input variables (and possibly a list of independent uniformly random keys when privacy is taken into account[15]); while a *linear decoding function* computes a linear combination of workers' output. We essentially need to prove that (a) given any multilinear f, any linear encoding scheme that achieves any (S, A, T) requires at least $N \geq (K + T - 1)\deg f + S + 2A + 1$ workers when $T > 0$ or $N \geq K\deg\, f - 1$, and $N \geq K(S + 2A + 1)$ workers in other cases; (b) for a general polynomial f, any scheme that uses linear encoding and decoding requires at least the same number of workers, if the characteristic of $\mathbb{F}$ is 0 or greater than deg f.

The proof rely on the following key lemma, which characterizes the *recovery threshold* of any encoding scheme, defined as the minimum number of workers that the master needs to wait to guarantee decodability.

Lemma 1. *Given any multilinear f, the recovery threshold of any valid linear encoding scheme, denoted by R, satisfies*

$$\begin{aligned} R \geq & R_{\text{LCC}}(N, K, f) \triangleq \\ & \min\{(K-1)\deg f + 1,\ N - \lfloor N/K \rfloor + 1\}. \quad (3) \end{aligned}$$

Moreover, if the encoding scheme is T private, we have $R \geq R_{\text{LCC}}(N, K, f) + T \cdot \deg f$.

The proof of Lemma 1 can be found in Appendix E, by constructing instances of the computation process for any assumed scheme that achieves smaller recovery threshold, and proving that such scheme fails to achieve decodability in these instances. Intuitively, note that the recovery threshold is exactly the difference between N and the number of stragglers that can be tolerated, inequality (3) in fact proves that LCC (described in Section 4 and Appendix G) achieves the optimum resiliency, as it exactly achieves the stated recovery threshold. Similarly, one can verify that Lemma 1 essentially states that LCC achieves the optimal tradeoff between resiliency and privacy.

Assuming the correctness of Lemma 1, the two parts of Theorem 2 can be proved as follows. To prove part (a) of the converses, we need to extend Lemma 1 to also take adversaries into account. This is achieved by using an extended concept of Hamming distance, defined in [Yu et al., 2018] for coded computing. Part (b) requires generalizing Lemma 1 to arbitrary polynomial functions, which is proved by showing that for any f that achieves any (S, T) pair, there exists a multilinear function with the same degree for which a computation scheme can be found to achieves the same requirement. The detailed proofs can be found in Appendices F and G respectively.

6 Application to Linear Regression and Experiments on AWS EC2

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

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 problem by iteratively moving the weight along the negative gradient direction, 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 comprising a master node and n worker nodes, we first partition $\boldsymbol{X} = [\boldsymbol{X}_1 \cdots \boldsymbol{X}_n]^\top$ into n sub-matrices. Each worker stores r coded sub-matrices generated from linearly combining $\boldsymbol{X}_j$s, for some parameter $1 \leq r \leq n$. Given the current weight $\boldsymbol{w}$, each worker performs computation using its local storage, and sends the result to the master. Master recovers $\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} = \sum_{j=1}^n \boldsymbol{X}_j\boldsymbol{X}_j^\top\boldsymbol{w}$ using the results from a subset of fastest workers.[16] To measure performance of any linear regression scheme, we consider the metric *recovery threshold* (denoted by

[15]This is well defined as we assumed that $\mathbb{V}$ is finite when $T > 0$.

[16]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.

R), defined as the minimum number of workers the master needs to wait for, to guarantee decodability (i.e., tolerating the remaining stragglers).

We cast this gradient computation to the computing model in Section 2, by grouping the sub-matrices into $K = \lceil \frac{n}{r} \rceil$ blocks such that $\boldsymbol{X} = [\bar{\boldsymbol{X}}_1 \cdots \bar{\boldsymbol{X}}_K]^\top$. Then computing $\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{w}$ reduces to computing 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 $\bar{\boldsymbol{X}}_1, \ldots, \bar{\boldsymbol{X}}_K$. Now, we can use LCC to decide on the coded storage as in (2), and achieve a recovery threshold of $R_{\text{LCC}} = 2(K-1)+1 = 2\lceil \frac{n}{r} \rceil - 1$ (Theorem 1).[17]

Comparisons with state of the arts. The conventional uncoded scheme picks $r = 1$, and has each worker j compute $\boldsymbol{X}_j \boldsymbol{X}_j^\top \boldsymbol{w}$. Master needs result from each work, yielding a recovery threshold of $R_{\text{uncoded}} = n$. By redundantly storing/processing $r > 1$ *uncoded* sub-matrices at each worker, the "gradient coding" (GC) methods [Tandon et al., 2017, Halbawi et al., 2017, Raviv et al., 2017] 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 [Lee et al., 2015] requires two rounds of computation. In the first round, an intermediate vector $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}$ is computed distributedly, which is re-distributed to the workers in the second round for them to collaboratively compute $\boldsymbol{X}^\top \boldsymbol{z}$. Each worker stores coded data generated using MDS codes from $\boldsymbol{X}$ and $\boldsymbol{X}^\top$ respectively. 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 at each worker is the same for GC and LCC, 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 whose sizes are respectively proportional to m and d, whereas each LCC worker only sends one dimension-d vector.

We run linear regression on AWS EC2 using Nesterov's accelerated gradient descent, where all nodes are implemented on `t2.micro` instances. We generate synthetic datasets of m data points, by 1) randomly sampling a true weight $\boldsymbol{w}^*$, 2) randomly sampling each input $\boldsymbol{x}_i$ of d features and computing its output $y_i = \boldsymbol{x}_i^\top \boldsymbol{w}^*$. For each dataset, we run GD for 100 iterations over $n = 40$ workers. We consider different dimensions of input matrix $\boldsymbol{X}$ as listed in the following scenarios.

[17]This recovery threshold is also optimum within a factor of 2, as we proved in Appendix J.

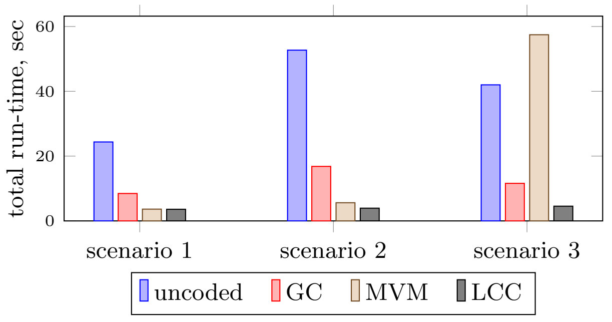

Figure 2: Run-time comparison of LCC with other three schemes: conventional uncoded, GC, and MVM.

- Scenario 1 & 2: $(m, d) = (8000, 7000)$.
- Scenario 3: $(m, d) = (160000, 500)$.

We let the system run with naturally occurring stragglers in scenario 1. To mimic the effect of 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 implement LCC, we set the β_i parameters to $1, \ldots, \frac{n}{r}$, and the α_i parameters to $0, \ldots, n-1$. To avoid numerical instability due to large entries of the decoding matrix, we can embed input data into a large finite field, and apply LCC in it with exact computations. However in all of 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 the run-time over the storage assigned between two rounds of matrix-vector multiplications. We plot the measured run-times in Figure 2, and list the detailed breakdowns of all scenarios in Appendix K.

We draw the following conclusions from 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 the number of inputs m is close to the number of features d, LCC achieves a similar performance as MVM. However, when we have much more data points in scenario 3, LCC finishes substantially faster than MVM by as much as 12.65×. The main reason for this subpar performance 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 are proportional to m). However, the amount of communication from each worker or master is proportional to d for all other schemes, which is much smaller than m in scenario 3.

Acknowledgement This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, ONR Award No. N00014-16-1-2189, 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

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

[Ben-Or et al., 1988] Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM.

[Berlekamp, 1968] Berlekamp, E. (1968). Nonbinary bch decoding (abstr.). *IEEE Transactions on Information Theory*, 14(2):242–242.

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

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

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

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

[Cramer et al., 2015] Cramer, R., Damgrd, I. B., and Nielsen, J. B. (2015). *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, New York, NY, USA, 1st edition.

[Dean and Barroso, 2013] Dean, J. and Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, 56(2):74–80.

[Dutta et al., 2016] Dutta, S., Cadambe, V., and Grover, P. (2016). Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2092–2100.

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

[Ezzeldin et al., 2017] Ezzeldin, Y. H., Karmoose, M., and Fragouli, C. (2017). Communication vs distributed computation: an alternative trade-off curve. *arXiv preprint arXiv:1705.08966*.

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

[Huang, 2017] Huang, W. (2017). *Coding for Security and Reliability in Distributed Systems*. PhD thesis, California Institute of Technology.

[Karakus et al., 2017] Karakus, C., Sun, Y., Diggavi, S., and Yin, W. (2017). Straggler mitigation in distributed optimization through data encoding. In *Advances in Neural Information Processing Systems*, pages 5440–5448.

[Kedlaya and Umans, 2011] Kedlaya, K. S. and Umans, C. (2011). Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802.

[Konstantinidis and Ramamoorthy, 2018] Konstantinidis, K. and Ramamoorthy, A. (2018). Leveraging Coding Techniques for Speeding up Distributed Computing. *ArXiv e-prints*.

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

[Lee et al., 2018] Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. (2018). Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529.

[Li et al., 2014a] Li, M., Andersen, D. G., Smola, A., and Yu, K. (2014a). 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, pages 19–27, Cambridge, MA, USA. MIT Press.

[Li et al., 2014b] Li, M., Andersen, D. G., Smola, A. J., and Yu, K. (2014b). Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27.

[Li et al., 2017a] Li, S., Kalan, S. M. M., Avestimehr, A. S., and Soltanolkotabi, M. (2017a). Near-optimal straggler mitigation for distributed gradient methods. *arXiv preprint arXiv:1710.09990*.

[Li et al., 2015] Li, S., Maddah-Ali, M. A., and Avestimehr, A. S. (2015). Coded MapReduce. In *Proceedings of the 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 964–971.

[Li et al., 2018a] Li, S., Maddah-Ali, M. A., Yu, Q., and Avestimehr, A. S. (2018a). A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128.

[Li et al., 2017b] Li, S., Supittayapornpong, S., Maddah-Ali, M. A., and Avestimehr, S. (2017b). Coded terasort. *IPDPSW*.

[Li et al., 2018b] Li, S., Yu, M., Avestimehr, S., Kannan, S., and Viswanath, P. (2018b). Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously. *arXiv preprint arXiv:1809.10361*.

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

[Massey, 1969] Massey, J. (1969). Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15(1):122–127.

[Mohassel and Zhang, 2017] Mohassel, P. and Zhang, Y. (2017). Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 19–38.

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

[Prakash et al., 2018] Prakash, S., Reisizadeh, A., Pedarsani, R., and Avestimehr, S. (2018). Coded computing for distributed graph analytics. *arXiv preprint arXiv:1801.05522*.

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

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

[Roth, 2006] Roth, R. (2006). *Introduction to coding theory*. Cambridge University Press.

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

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Commun. ACM*, 22(11):612–613.

[So et al., 2019] So, J., Guler, B., Avestimehr, A. S., and Mohassel, P. (2019). Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. *arXiv preprint arXiv:1902.00641*.

[Tandon et al., 2017] Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. (2017). 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*, pages 3368–3376.

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

[Yadwadkar et al., 2016] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., and Katz, R. (2016). Multi-task learning for straggler avoiding predictive job scheduling. *Journal of Machine Learning Research*, 17(106):1–37.

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

[Yu et al., 2017a] Yu, Q., Li, S., Maddah-Ali, M. A., and Avestimehr, A. S. (2017a). How to optimally allocate resources for coded distributed computing? In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7.

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

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