ϵ-Approximate Coded Matrix Multiplication is Nearly Twice as Efficient as Exact Multiplication

Haewon Jeong, *Member, IEEE*, Ateet Devulapalli, Viveck R. Cadambe, *Member, IEEE*, Flavio P. Calmon, *Member, IEEE*

***Abstract*—We study coded distributed matrix multiplication from an approximate recovery viewpoint. We consider a system of P computation nodes where each node stores $1/m$ of each multiplicand via linear encoding. Our main result shows that the matrix product can be recovered with ϵ relative error from any m of the P nodes for any $\epsilon > 0$. We obtain this result through a careful specialization of MatDot codes — a class of matrix multiplication codes previously developed in the context of exact recovery ($\epsilon = 0$). Since prior results showed that MatDot codes achieve the best exact recovery threshold for a class of linear coding schemes, our result shows that allowing for mild approximations leads to a system that is nearly twice as efficient as exact reconstruction. For Entangled-Poly codes — which are generalizations of MatDot codes — we show that approximation reduces the recovery threshold from p^2q+q-1 to p^2q, when the input matrices $\mathbf{A}, \mathbf{B}$ are split respectively in to a $p \times q$ and $q \times p$ grids of equal-sized submatrices.**

***Index Terms*—Coded computing, Distributed computing, Fault-tolerant computing, Error correction codes, Approximate algorithms, Matrix Multiplication, Distributed machine learning**

I. Introduction

Coded computing has emerged as a promising paradigm to resolving straggler and security bottlenecks in large-scale distributed computing platforms [1]–[24]. The foundations of this paradigm lie in novel code constructions for elemental computations such as matrix operations and polynomial computations, and fundamental limits on their performance. In this paper, we show that the state-of-the-art fundamental limits for such elemental computations grossly underestimate the performance by focusing on *exact* recovery of the computation output. By allowing for mild *approximations* of the computation output, we demonstrate significant improvements in terms of the trade-off between fault-tolerance and the degree of redundancy.

Consider a distributed computing system with P nodes for performing the matrix multiplication $\mathbf{AB}$. If each node is required to store a fraction $1/m$ of both matrices, the best known recovery threshold is equal to $2m-1$ achieved by the MatDot code [3]. Observe the contrast between distributed coded *computation* with distributed data *storage*, where a maximum distance separable (MDS) code ensures that if each node stores a fraction $1/m$ of the data, then the data can be recovered from any m nodes[1] [25], [26]. Indeed, the recovery threshold of m is crucial to the existence of practical codes that bring fault-tolerance to large-scale data storage systems with relatively minimal overheads (e.g., single parity and Reed-Solomon codes [27]).

The contrast between data storage and computation is even more pronounced when we consider the generalization of matrix-multiplication towards multi-variate polynomial evaluation $f(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_\ell)$ where each node is allowed to store a fraction $1/m$ of each of $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_\ell$. In this case, the technique of Lagrange coded-computing [5] demonstrates that the recovery threshold is $d(m-1)+1$, where d is the degree of the polynomial. Note that a recovery threshold of m is only obtained for the special case of degree $d=1$ polynomials. The case of $d=1$ i.e., elementary linear transformations that were originally studied in [28][2] can . While the results of [3], [29] demonstrate that the amount of redundancy is much less than previously thought for degree $d>1$ computations, these codes still require an overwhelming amount of additional redundancy—even

This material is based upon work supported by the National Science Foundation under grants CIF 1900750, CAREER 1845852, IIS 1926925, and CCF 1763657.

Haewon Jeong and Flavio Calmon are with the John A. Paulson School of Engineering and Applied Sciences at Harvard University. E-mails: haewon, flavio@seas.harvard.edu.

Ateet Devulapalli and Viveck R. Cadambe are with the School of Electrical Engineering and Computer Science at Pennsylvania State University. E-mails: azd565,viveck@psu.edu

[1]This essentially translates to the Singleton bound being tight for a sufficiently large alphabet

[2]The case of distributed storage can be viewed as the case of $d=1$; specifically, we may view the problem of recovering the data matrix $\mathbf{A}$ as equivalent to recovering linear transformation $\mathbf{A}$ applied to all columns of the identity matrix, hence the recovery threshold of m.

to tolerate a single failed node—when compared to codes for distributed storage.

A. Summary of Results

Our paper is the result of the search for an analog of MDS codes—in terms of the amount of redundancy required—for coded-computation of polynomials with degree greater than 1. We focus on the case of coded matrix multiplication where the goal is to recover the matrix product $\mathbf{C} = \mathbf{AB}$. We consider a distributed computation system of P worker nodes similar to [2], [3]; we allow each worker to store an m-th fraction of matrices of $\mathbf{A}$, $\mathbf{B}$ via linear transformations (encoding). The workers output the product of the encoded matrices. A central master/fusion node collects the output of a set $\mathcal{S}$ of non-straggling workers and aims to decode $\mathbf{C}$ with a relative error of ϵ. The recovery threshold $K(m, \epsilon)$ is the cardinality of the largest minimal subset $\mathcal{S}$ that allows for such recovery. It has been shown in [3], [29] that, for natural classes of linear encoding schemes, $K(m, 0) = 2m - 1$.

Our main result shows that the MatDot code with a specific set of evaluation points is able to achieve $K(m, \epsilon) = m$, remarkably, for *any* $\epsilon > 0$. A simple converse shows that our result is tight. Our results mirrors several results in classical information theory (e.g., almost lossless data compression), where allowing ϵ-error for any $\epsilon > 0$ leads to surprisingly significant improvements in performance. We also show that for PolyDot/Entangled polynomial codes [3], [29], [30] where matrices $\mathbf{A}, \mathbf{B}$ are restricted to be split as $p \times q$ and $q \times p$ block matrices respectively, we improve the recovery threshold[3] from $p^2q + q - 1$ to p^2q by allowing ϵ-error. When we fix the splitting of input matrices as in PolyDot codes, the gain in recovery threshold we obtain from approximation is not 2x as in ϵ-MatDot codes. ϵ-Approximate MatDot codes are a special case of ϵ-Approximate PolyDot codes with $p = 1$, which has the biggest relative gain compared to the exact computation counterpart. We believe that these results open up a new avenue in coded computing research via revisiting existing code constructions and allowing for an ϵ-error.

Through an application of our code constructions to distributed training for classification via logistic regression, we show that our approximations suffices to obtain accurate classification results in practice.

B. Related Work

The study of coded computing for elementary linear algebra operations, starting from [4], [28], is an active research area (see surveys [22]–[24]). Notably, the recovery thresholds for matrix multiplication were established via achievability and converse results respectively in [2], [3], [29]. The Lagrange coded computing framework of [5] generalized the systematic MatDot code construction of [3] to the context of multi-variate polynomial evaluations and established a tight lower bound on the recovery threshold. These works focused on exact recovery of the computation output.

References [31]–[33] studied the idea of gradient coding from an approximation viewpoint, and demonstrated improvements in recovery threshold over exact recovery. However, in contrast with our results, the error obtained either did not correct all possible error patterns with a given recovery threshold (i.e., they considered a probabilistic erasure model), and the relative error of their approximation was lower bounded. The references that are most relevant to our work are [20], [21], [34], which also aim to improve the recovery threshold of coded matrix multiplication by allowing for a relative error of ϵ. These references use random linear coding (i.e., sketching) techniques to obtain a recovery threshold $\overline{K}(\epsilon, \delta, m)$ where δ is the probability of failing to recover the matrix product with a relative error of ϵ; the problem statement of [34] is particularly similar to ours. Our results can be viewed as a strict improvement over this prior work, as we are able to obtain a recovery threshold of m even with $\delta = 0$, whereas the recovery threshold is at least $2m - 1$ for $\delta = 0$ in [20], [21], [34].

A related line of work in [35], [36] study coded polynomial evaluation beyond exact recovery and note techniques to improve the quality of the approximation. References [37], [38] develops machine learning techniques for approximate learning; while they show empirical existence codes with low recovery thresholds (such as single parity codes [37]) for learning tasks they do not provide theoretical guarantees. Specifically, while [37], [38] shows the benefits of approximation in terms of recovery threshold, it is unclear whether these benefits appear in their scheme due to the special structure of the data, or whether the developed codes work for all realizations of the data. In contrast with [35]–[38], we are the first to establish the strict gap in the recovery thresholds for ϵ-error computations versus exact computation for matrix multiplication, which is a canonical case of degree 2 polynomial evaluation.

A tangentially related body of work [39]–[42] studies the development of numerically stable coded computing techniques. While some of these works draw on techniques from approximation theory, they focus on maintaining recovery threshold the same as earlier constructions, but bounding the approximation error of the

[3]Strictly speaking, the recovery threshold of entangled polynomial codes depends on the bilinear complexity, which can be smaller than $p^2q + q - 1$ [29].

output in terms of the precision of the computation.

II. System Model and Problem Statement

A. Notations

We define $[n] \triangleq \{1, 2, \cdots, n\}$. We use bold fonts for vectors and matrices. $A[i, j]$ denotes the (i, j)-th entry of an $M \times N$ matrix $\mathbf{A}$ ($i \in [M], j \in [N]$) and $v[i]$ is the i-th entry of a length-N vector $\mathbf{v}$ ($i \in [N]$).

B. System Model

We consider a distributed computing system with a master node and P worker nodes. At the beginning of the computation, a master node distributes appropriate tasks and inputs to worker nodes. Worker nodes perform the assigned task and send the result back to the master node. Worker nodes are prone to failures or delay (stragglers). Once the master node receives results from a sufficient number of worker nodes, it produces the final output.

We are interested in distributed matrix multiplication, where the goal is to compute

$$\mathbf{C} = \mathbf{AB}. \tag{1}$$

We assume $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ are matrices with a bounded norm, i.e.,

$$||\mathbf{A}||_F \leq \eta \quad \text{and} \quad ||\mathbf{B}||_F \leq \eta, \tag{2}$$

where $||\cdot||_F$ denotes Frobenius norm. We further assume that worker nodes have memory constraints such that each node can hold only an m-th fraction of $\mathbf{A}$ and an m-th fraction of $\mathbf{B}$ in memory. To meet the memory constraint, we divide $\mathbf{A}, \mathbf{B}$ into small equal-sized sub-blocks as follows[4]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,q} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{p,1} & \cdots & \mathbf{A}_{p,q} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,p} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{q,1} & \cdots & \mathbf{B}_{q,p} \end{bmatrix}, \tag{3}$$

where $pq = m$. When $p = 1$, we simply denote

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \ldots & \mathbf{A}_m \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_m \end{bmatrix}.$$

To mitigate failures or stragglers, a master node encodes redundancies through linear encoding. The i-th worker node receives encoded inputs $\widetilde{\mathbf{A}}_i$ and $\widetilde{\mathbf{B}}_i$ such that:

$$\widetilde{\mathbf{A}}_i = f_i(\mathbf{A}_{1,1}, \cdots, \mathbf{A}_{p,q}), \quad \widetilde{\mathbf{B}}_i = g_i(\mathbf{B}_{1,1}, \cdots, \mathbf{B}_{q,p}),$$

[4]We limit ourselves to splitting the input matrices into a grid of submatrices. Splitting into an arbitrary shape is beyond the scope of this work.

where

$$f_i : \underbrace{\mathbb{R}^{\frac{n}{p} \times \frac{n}{q}} \times \cdots \times \mathbb{R}^{\frac{n}{p} \times \frac{n}{q}}}_{pq=m} \to \mathbb{R}^{\frac{n}{p} \times \frac{n}{q}}, \tag{4}$$

$$g_i : \underbrace{\mathbb{R}^{\frac{n}{q} \times \frac{n}{p}} \times \cdots \times \mathbb{R}^{\frac{n}{q} \times \frac{n}{p}}}_{m} \to \mathbb{R}^{\frac{n}{q} \times \frac{n}{p}}. \tag{5}$$

We assume that f_i, g_i are linear, i.e., their outputs are linear combinations of m inputs. For example, we may have $f_i(\mathbf{Z}_1, \cdots, \mathbf{Z}_m) = \gamma_{i,1}\mathbf{Z}_1 + \cdots + \gamma_{i,m}\mathbf{Z}_m$ for some $\gamma_{i,j} \in \mathbb{R}$ ($j \in [m]$). .

Worker nodes are oblivious of the encoding/decoding process and simply perform matrix multiplication on the inputs they receive. In our case, each worker node computes

$$\widetilde{\mathbf{C}}_i = \widetilde{\mathbf{A}}_i \widetilde{\mathbf{B}}_i, \tag{6}$$

and returns the $\frac{n}{p} \times \frac{n}{p}$ output matrix $\widetilde{\mathbf{C}}_i$ to the master node.

Finally, when the master node receives outputs from a subset of worker nodes, say $\mathcal{S} \subseteq [P]$, it performs decoding:

$$\widehat{\mathbf{C}}_{\mathcal{S}} = d_{\mathcal{S}}((\widetilde{\mathbf{C}}_i)_{i \in \mathcal{S}}), \tag{7}$$

where $\{d_{\mathcal{S}}\}_{\mathcal{S} \subseteq [P]}$ is a set of predefined decoding functions that take $|\mathcal{S}|$ inputs from $\mathbb{R}^{\frac{n}{p} \times \frac{n}{p}}$ and outputs an n-by-n matrix. Note that we do not restrict the decoders $d_{\mathcal{S}}$ to be linear.

C. Approximate Recovery Threshold

Let $\mathbf{f}$ and $\mathbf{g}$ be vectors of linear encoding functions:

$$\mathbf{f} = \begin{bmatrix} f_1 & \cdots & f_P \end{bmatrix}, \mathbf{g} = \begin{bmatrix} g_1 & \cdots & g_P \end{bmatrix},$$

and let $\mathbf{d}$ be a length-2^P vector of decoding functions $d_{\mathcal{S}}$ for all subsets $\mathcal{S} \subseteq [P]$. More specifically, $d_{\mathcal{S}}$ is a decoding function for the scenario where worker nodes in set $\mathcal{S}$ are successful in returning their computations to the master node and all other worker nodes fail. We say that the ϵ-approximate recovery threshold of $\mathbf{f}, \mathbf{g}, \mathbf{d}$ is K if for any $\mathbf{A}$ and $\mathbf{B}$ that satisfy the norm constraints (2), the decoded matrix satisfies

$$|\widehat{C}_{\mathcal{S}}[i, j] - C[i, j]| \leq \epsilon \quad (i, j \in [n]) \tag{8}$$

for every $\mathcal{S} \subseteq [P]$ such that $|\mathcal{S}| \geq K$. We denote this recovery threshold as $K(p, q, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d})$. Moreover, let $K^*(p, q, \epsilon)$ be defined as the minimum of $K(p, q, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d})$ over all possible linear encoding functions $\mathbf{f}$, $\mathbf{g}$ and all possible decoding functions $\mathbf{d}$, i.e.,

$$K^*(p, q, \epsilon) \triangleq \min_{\mathbf{f}, \mathbf{g}, \mathbf{d}} K(p, q, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d}). \tag{9}$$

Furthermore, we define $K^*(m, \epsilon)$ as the minimum over all combinations of p, q such that $pq = m$.

D. Existing code constructions for exact recovery ($\epsilon = 0$)

We provide a brief description on previous works on exact recovery codes for distributed matrix multiplication. First, we introduce MatDot codes where $p = 1$ and $q = m$.

Construction 1 (MatDot Codes [3])**.** *Define polynomials $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ as follows:*

$$p_{\mathbf{A}}(x) = \sum_{i=1}^{m} \mathbf{A}_i x^{i-1}, p_{\mathbf{B}}(x) = \sum_{j=1}^{m} \mathbf{B}_j x^{m-j}. \quad (10)$$

Let $\lambda_1, \lambda_2, \ldots, \lambda_P$ be P distinct elements in $\mathbb{R}$. The i-th worker receives encoded versions of matrices:

$$\widetilde{\mathbf{A}}_i = p_{\mathbf{A}}(\lambda_i) = \mathbf{A}_1 + \lambda_i \mathbf{A}_2 + \cdots + \lambda_i^{m-1} \mathbf{A}_m,$$
$$\widetilde{\mathbf{B}}_i = p_{\mathbf{B}}(\lambda_i) = \mathbf{B}_m + \lambda_i \mathbf{B}_{m-1} + \cdots + \lambda_i^{m-1} \mathbf{B}_1,$$

and then computes matrix multiplication on the encoded matrices:

$$\widetilde{\mathbf{C}}_i = \widetilde{\mathbf{A}}_i \widetilde{\mathbf{B}}_i = p_{\mathbf{A}}(\lambda_i) p_{\mathbf{B}}(\lambda_i) = p_{\mathbf{C}}(\lambda_i).$$

The polynomial $p_{\mathbf{C}}(x)$ has degree $2m - 2$ and has the following form:

$$p_{\mathbf{C}}(x) = \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{A}_i \mathbf{B}_j x^{m-1+(i-j)}. \quad (11)$$

Once the master node receives outputs from $2m - 1$ successful worker nodes, it can recover the coefficients of $p_{\mathbf{C}}(x)$ through polynomial interpolation, and then recover $\mathbf{C} = \sum_{i=1}^{m} \mathbf{A}_i \mathbf{B}_i$ as the coefficient of x^{m-1} in $p_{\mathbf{C}}(x)$. □

The recovery threshold of MatDot codes is $2m - 1$ because the polynomial $p_{\mathbf{C}}(x)$ is a degree-$(2m - 2)$ polynomial and we need $2m - 1$ points to recover all of the coefficients of $p_{\mathbf{C}}(x)$. However, in order to recover $\mathbf{C}$, we only need the coefficient of x^{m-1} in $p_{\mathbf{C}}(x)$. Through an achievability scheme in [3] and a converse in [29], for exact recovery, the optimal threshold has been characterized to be $2m - 1$:

Theorem 1 (Adaptation of Theorem 2 in [29] and Theorem III.1 in [3])**.** *Under the system model given in Section II-B*

$$K^*(m, \epsilon = 0) = 2m - 1. \quad (12)$$

Next, we describe PolyDot (Entangled-Poly) codes which are a generalization of MatDot codes for arbitrary p and q.

Construction 2 (PolyDot (Entangled-Poly) Codes [3], [29])**.** *In [3], a general framework for PolyDot codes is proposed as follows:*

$$p_{\mathbf{A}}(x, y) = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{A}_{i,j} x^{i-1} y^{j-1},$$
$$p_{\mathbf{B}}(y, z) = \sum_{k=1}^{q} \sum_{l=1}^{p} \mathbf{B}_{k,l} y^{q-k} z^{l-1},$$

where the input matrices are split as (3)*. Substituting $x = y^q$ and $z = y^{pq}$ results in Entangled-Poly codes [29]:*

$$p_{\mathbf{A}}(y) = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{A}_{i,j} y^{q(i-1)+(j-1)},$$
$$p_{\mathbf{B}}(y) = \sum_{k=1}^{q} \sum_{l=1}^{p} \mathbf{B}_{k,l} y^{q-k+pq(l-1)}.$$

In the product polynomial $p_{\mathbf{C}}(y) = p_{\mathbf{A}}(y) p_{\mathbf{B}}(y)$, the coefficient of $y^{(i-1)q+q-1+pq(l-1)} = y^{iq+pq(l-1)-1}$ is $\mathbf{C}_{i,l} = \sum_{k=1}^{q} \mathbf{A}_{i,k} \mathbf{B}_{k,l}$. The degree of $p_{\mathbf{C}}$ is:

$$(p-1)q + (q-1) + (q-1) + pq(p-1) = p^2 q + q - 2.$$

Hence, a recovery threshold of $p^2 q + q - 1$ is achievable.

E. Main Result

Our main result is summarized in the below theorem:

Main Theorem. *Under the system model given in Section II-B, the optimal ϵ-approximate recovery threshold for arbitrary p and q such that $pq = m$ is:*

$$K^*(m, \epsilon) = m. \quad (13)$$

(Achievability – Corollary 1)
For any $\epsilon > 0$, the ϵ-approximate MatDot codes in Construction 4 achieves:

$$K(1, m, \epsilon, \mathbf{f}_{\epsilon\text{-MatDot}}, \mathbf{g}_{\epsilon\text{-MatDot}}, \mathbf{d}_{\epsilon\text{-MatDot}}) = m.$$

(Converse – Corollary 2)
For all $0 < \epsilon < \frac{\eta^2}{n}$, $K^*(m, \epsilon) \geq m$.

This shows that for a given storage constraint m, the optimal approximate recovery threshold for any $\epsilon > 0$ is m. As compared to the exact recovery threshold of $2m - 1$, this is almost 2x gain. The main result is obtained as a special case from our more general result on PolyDot (Entangled-Poly) codes:

(Achievability – Theorem 2)
For any $\epsilon > 0$, the ϵ-approximate PolyDot codes in Construction 3 achieves:

$$K(p, q, \epsilon, \mathbf{f}_{\epsilon-\text{PolyDot}}, \mathbf{g}_{\epsilon\text{-PolyDot}}, \mathbf{d}_{\epsilon\text{-PolyDot}}) = p^2 q.$$

(Lower Bound – Theorem 3)
For all $0 < \epsilon < \frac{\eta^2}{n \cdot \min(p,q)}$, $K(p, q, \epsilon) \geq pq$.

Note that for the general case, the gain in recovery threshold is from $p^2q + q - 1$ to p^2q. Furthermore, the gap between the achievable scheme and the lower bound is tight only when $p = 1$ and $q = m$, i.e., for the case of Approximate MatDot codes.

III. Theoretical Characterization of $K^*(m, \epsilon)$

In this section, we derive the main theorem given in Section II-E. We first propose the construction of ϵ-approximate PolyDot codes that can achieve the recovery threshold of p^2q for ϵ approximation error for any $\epsilon > 0$. Then, we give a lower bound on the approximate recovery threshold for sufficiently small ϵ. By substituting $p = 1, q = m$, we obtain the result for Approximate MatDot codes, which meets the lower bound m. Finally, we conclude the section by providing an insight of the proposed approximate code constructions.

A. Approximate PolyDot Codes

Recall that for arbitrary p and q, the recovery threshold of PolyDot codes (Entangled-Poly codes) is $p^2q + q - 1$. We propose ϵ-approximate PolyDot codes which achieves the ϵ-approximate recovery threshold of p^2q by choosing evaluation points close to 0.

Construction 3 (ϵ-Approximate PolyDot codes)**.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be matrices in* $\mathbb{R}^{n\times n}$ *that satisfy* $||\mathbf{A}||_F, ||\mathbf{B}||_F \leq \eta$ *and let* $\epsilon > 0$ *be a constant. Then,* ϵ*-Approximate PolyDot code is a PolyDot code defined in Construction 2 with evaluation points* $\lambda_1, \ldots, \lambda_P$ *that satisfy:*

$$|\lambda_i| < \min\left(\frac{\epsilon}{\eta^2 q(p^2q-1)}, \frac{1}{p^2q-1}\right), \quad i \in [P]. \quad (14)$$

The following theorem states that the recovery threshold can be reduced by $q - 1$ by allowing ϵ-approximate recovery.

Theorem 2 (Achievability)**.** *For any* $\epsilon > 0$*, the* ϵ*-approximate PolyDot codes in Construction 3 achieves:*

$$K(p, q, \epsilon, \mathbf{f}_{\epsilon\text{-PolyDot}}, \mathbf{g}_{\epsilon\text{-PolyDot}}, \mathbf{d}_{\epsilon\text{-PolyDot}}) = p^2 q,$$

where $\mathbf{f}_{\epsilon\text{-PolyDot}}, \mathbf{g}_{\epsilon\text{-PolyDot}}, \mathbf{d}_{\epsilon\text{-PolyDot}}$ *are encoding and decoding functions specified by Construction 3.*

We defer the full proof to Appendix A.

Theorem 2 shows that for *any* matrices $\mathbf{A}$ and $\mathbf{B}$, and for any set of p^2q successful nodes, ϵ-approximate PolyDot codes can recover the output with error at most ϵ. We now show that if we only have $pq - 1$ successful nodes, there exist matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ such that the recovery error cannot be made arbitrarily small for any encoding scheme.

Theorem 3 (Lower bound on recovery threshold K for arbitrary p and q)**.** *Under the system model given in Section II, for any* $0 < \epsilon < \frac{\eta^2}{n \cdot \min(p,q)}$*,*

$$K(p, q, \epsilon) \geq pq. \quad (15)$$

Proof Sketch. To prove by contradiction, we construct matrices $\mathbf{A}$ and $\mathbf{B}$ that lead to large reconstruction error. that the error is big. To do that, we realize that the encoding coefficients form an underdetermined system of equations and we use vector from its null space to generate a matrix $\mathbf{A}$ such that the encoded output is $\mathbf{0}$. We then use the properties of Frobenius norm to argue that the decoding error is lower bounded. Full proof is given in Appendix B. □

B. Approximate MatDot Codes

Observe that PolyDot codes are a general version of MatDot codes, i.e. by setting $p = 1, q = m$. So, substituting $p = 1, q = m$ in Construction 3, Theorem 2 and Theorem 3 we obtain:

Construction 4 (ϵ-Approximate MatDot codes)**.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be matrices in* $\mathbb{R}^{n\times n}$ *that satisfy* $||\mathbf{A}||_F, ||\mathbf{B}||_F \leq \eta$*. Let* $\epsilon \in \mathbb{R}$ *be a constant. Then,* ϵ*-Approximate MatDot code is a MatDot code defined in Construction 1 with evaluation points* $\lambda_1, \ldots, \lambda_P$ *that satisfy:*

$$|\lambda_i| < \min\left(\frac{\epsilon}{\eta^2 \cdot m(m-1)}, \frac{1}{m-1}\right), \quad i \in [P]. \quad (16)$$

We then show that this construction has the approximate recovery threshold of m.

Corollary 1 (Achievability)**.** *For any* $\epsilon > 0$*, the* ϵ*-Approximate MatDot codes in Construction 4 achieves:*

$$K(1, m, \epsilon, \mathbf{f}_{\epsilon\text{-MatDot}}, \mathbf{g}_{\epsilon\text{-MatDot}}, \mathbf{d}_{\epsilon\text{-MatDot}}) = m, \quad (17)$$

where $\mathbf{f}_{\epsilon\text{-MatDot}}, \mathbf{g}_{\epsilon\text{-MatDot}}, \mathbf{d}_{\epsilon\text{-MatDot}}$ *are encoding and decoding functions specified by Construction 4.*

Corollary 2 (Converse)**.** *Under the system model given in Section II, for any* $0 < \epsilon < \frac{\eta^2}{n}$*,*

$$K^*(m, \epsilon) \geq m. \quad (18)$$

Remark 1. *The error bound provided by Theorem 2 and Corollary 1 is an absolute bound, i.e.,* $||\widehat{\mathbf{C}} - \mathbf{C}||_{\max} \leq \epsilon$.

This is because we choose the evaluation points which are scaled by η, which is the upper bound of $||\mathbf{A}||_F$ and $||\mathbf{B}||_F$ as given in (2). If we do not assume prior knowledge on the upper bound of $||\mathbf{A}||_F$ and $||\mathbf{B}||_F$, we can choose λ_i's to be some small numbers, e.g., $|\lambda_i| \leq \Delta$, and then the error bound will be a relative bound on $\frac{||\widehat{\mathbf{C}}-\mathbf{C}||_{\max}}{||\mathbf{A}||_F||\mathbf{B}||_F}$. Furthermore, note that the bound on the max norm can be easily converted to bounds on other types of norm (e.g., Frobenius norm or 2-norm) within a constant factor using matrix norm equivalence relations.

C. An insight behind Approximate MatDot Codes

Recall that in MatDot codes the decoder receives evaluations of a degree $2m-1$ polynomial, and aims to recover the coefficient corresponding to x^{m-1}. The key idea of Approximate MatDot Codes to use MatDot codes, but to select evaluation points close to 0, specifically, in an interval around 0 that is proportional to ϵ. We below explain why the coefficient x^{m-1} can be *approximately* recovered for an arbitrary degree $2m-1$ polynomial with only m evaluations.

Let $S(x)$ be a polynomial of degree $2m-1$ and let $P(x)$ be a polynomial of degree m. Then, $S(x)$ can be written as:

$$S(x) = P(x)Q(x) + R(x), \tag{19}$$

and the degree of Q and R are both at most $m-1$. Now, let $\lambda_1, \ldots, \lambda_m$ be the roots of $P(x)$. Then,

$$S(\lambda_i) = R(\lambda_i). \tag{20}$$

If we have m evaluations at these points, we can exactly recover the coefficients of the polynomial $R(x)$.

Recall that we only need the coefficient of x^{m-1} in MatDot codes. Letting $P(x) = x^m$, $S(x)$ can be written as:

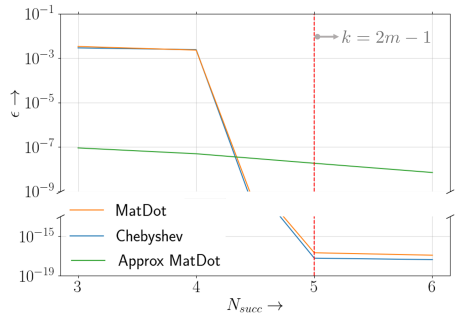
$$S(x) = x^m Q(x) + R(x). \tag{21}$$

Since the lower order terms are all in $R(x)$, the coefficient of x^{m-1} in $R(x)$ is equal to the coefficient of x^{m-1} in $S(x)$. Thus, recovering the coefficients of $R(x)$ is sufficient for MatDot decoding. However, x^m has only one root, 0. For approximate decoding, we can use points close to 0 as evaluation points to make $x^m \approx 0$. Then, we have:

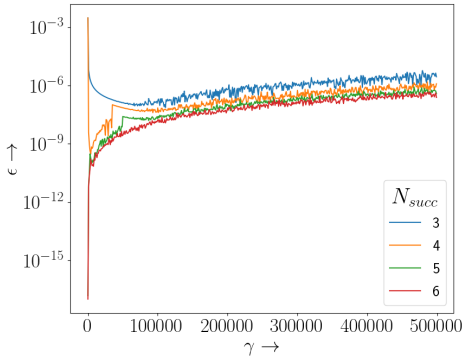
$$S(\lambda_i) = \lambda_i^m Q(\lambda_i) + R(\lambda_i) \approx R(\lambda_i). \tag{22}$$

When $|S(\lambda_i) - R(\lambda_i)|$ is small, we can use m evaluations of $S(\lambda_i)$'s to approximately interpolate $R(x)$. Moreover, when λ_i is small, we can also bound $|S(\lambda_i) - R(\lambda_i)|$ when Q has a bounded norm. In our case, S has a bounded norm due the norm constraints (2) on the input matrices and, thus, Q must have a bounded norm since the higher-order terms in S are solely determined by Q.

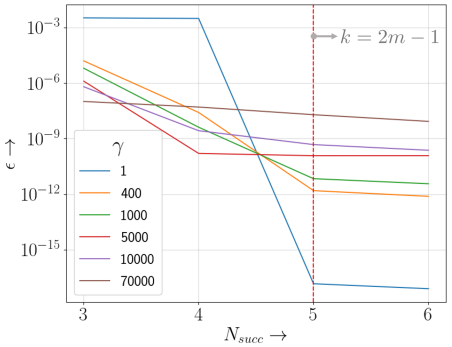
D. Approx MatDot Experimental results



(a) ϵ vs N_{succ} for $m = 3, P = 6$



(b) MatDot: ϵ vs γ for $m = 3, P = 6$



(c) MatDot: ϵ vs N_{succ} for $m = 3, P = 6$

Figure 1: Performance comparison between different coding methods, over various number of successful nodes for fixed $m = 3$ and $P = 6$ is shown in (a). In (b),(c) performance of approx MatDot codes are shown as the evaluation points decrease in magnitude. The y-axis represents the empirical evaluation of ϵ, i.e., $||\widehat{\mathbf{C}} - \mathbf{C}||_{\max}$.

In Fig. 1, we compare the performance of the conventional MatDot and Chebyshev polynomial-based codes [39] with the approximate MatDot codes developed in this paper. We show the error in the decoded matrix product, i.e., $\epsilon = ||\mathbf{C} - \widehat{\mathbf{C}}||_{\max}$. To compute this, we performed multiplications of two random matrices of

sizes 100×100, which were normalized to have unit Frobenius norm ($\eta = 1$). Each element in the matrices before normalization is a Gaussian random variable with PDF $\mathcal{N}(0, 1)$.

In the figures, the parameter N_{succ} represents the number of non-straggling nodes. Note the difference between N_{succ} and k; k represents the recovery threshold that the codes have been designed for, while N_{succ} represents the number of nodes that do not fail and is independent of code design. For example, for MatDot codes $k = 2m-1$ and for approximate MatDot codes $k = m$. Note that when N_{succ} out of P nodes fail, there are $\binom{P}{N_{\text{succ}}}$ different failure patterns. The decoding error of each failure pattern need not be equal and in practice we observed the same. Fig. 1, plots ϵ, which is the failure pattern that produced maximum decoding error.

The $2m - 1$ recovery threshold is highlighted in a red dotted vertical line for reference. MatDot and Approximate MatDot codes are constructed using the evaluation points $\boldsymbol{\lambda}(1)$ and $\boldsymbol{\lambda}(70000)$ respectively, where

$$\boldsymbol{\lambda}(\gamma) = \left\{ \frac{1}{\gamma} \cos\left(\frac{(2i-1)\pi}{2P} \right) \right\}_{i=1}^{P}.$$

The above equation is consistent with picking of Chebyshev nodes as our evaluation points. The Chebyshev nodes are a popular choice [43] to mitigate the well-known Runge phenomenon, where the interpolation error increases closer to the boundaries of the interval $[-1/\gamma, 1/\gamma]$. It is also instructive to note that the only difference between MatDot and Approx Matdot is the choice of γ, the encoding scheme is same.

Figure 1a demonstrates that for $N_{\text{succ}} \geq k = 2m - 1$, the Matdot and Chebyshev codes have very small error (10^{-16}), however, for $N_{\text{succ}} < k$ Approximate MatDot codes outperform Chebyshev and MatDot codes.

Figures 1b and 1c represent MatDot codes (with evaluation points $\boldsymbol{\lambda(\gamma)}$) behavior for increasing condition number (controlled with γ parameter). Observe that the error ϵ is composed of two quantities: ϵ_1, the interpolation error under infinite precision, and ϵ_2 the computation error due to finite precision. For $N_{\text{succ}} \geq k = 2m - 1$, $\epsilon_1 = 0$, ϵ_2 increases as γ. However, for $N_{\text{succ}} \leq k = 2m - 1$, ϵ_1 decreases and ϵ_2 increases, as γ increases; therefore in Fig 1b the error is non-monotonic in γ for $N_{\text{succ}} \leq k = 2m - 1$. Showing a different view, in Fig 1c, observe that for $N_{\text{succ}} \geq k = 2m - 1$ smaller γ has better loss, while the opposite behaviour is seen for $N_{\text{succ}} \leq k = 2m - 1$. The source code for Figure 1 is in [44].

IV. Application To Machine Learning

In this section, we illustrate that approximate coded computing is particularly useful for training machine learning (ML) models. ML models are usually trained using optimization algorithms that have inherent stochasticity (e.g., stochastic gradient descent). These algorithms are applied to finite, noisy training data. Consequently, ML models can be tolerant of the accuracy loss resulting from approximate computations during training. In fact, this loss can be insignificant when compared to other factors that impact training performance (parameter initialization, learning rate, dataset size, etc.).

We illustrate this point by considering a simple logistic regression training scenario modified to use coded computation. First, we describe how coded matrix multiplication strategies can be applied to training a logistic regression model. Then, we train a model on the MNIST dataset [45] using approximate coded computing strategies and show that the accuracy loss due to approximate coded matrix multiplication is very small.

A. Logistic regression model with coded computation

We consider logistic regression with cross entropy loss and softmax function. We identify parts of training steps where coded computation could be applied.

Consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1) \dots (\boldsymbol{x}_D, \boldsymbol{y}_D)\}$ and the loss function $L(\boldsymbol{W}; \mathcal{D})$ with gradient $\frac{\partial}{\partial \boldsymbol{W}} L(\boldsymbol{W}; \mathcal{D})$, for model $\boldsymbol{W}$. Let there be J classes in the dataset, and $\{\boldsymbol{y}_i\}_{i=1}^{D}$ be a set of one-hot encoded vectors, such that $y_{ji} = 1$ means i^{th} data point $\boldsymbol{x}_i$ belongs to j^{th} class. Let $\boldsymbol{Y} \in \mathbb{F}_2^{J \times D} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_D]$. Let $\boldsymbol{W} = [\boldsymbol{w}_1; \dots; \boldsymbol{w}_J]$ ($\boldsymbol{w}_j$ is a row vector) be a matrix that comprises the logistic regression training parameters. The cross entropy loss is given by:

$$L(\boldsymbol{W}; \mathcal{D}) = \sum_{i=1}^{D} \sum_{j=1}^{J} y_{ji} \log p(y_{ji} = 1 | \boldsymbol{x}_i) \tag{23}$$

where

$$p(y_{ji} = 1 | \boldsymbol{x}_i) = \text{softmax}(z_{ji}) = \frac{e^{z_{ji}}}{\sum_{j=1}^{J} e^{z_{ji}}}, \quad z_{ji} = \boldsymbol{w}_j \boldsymbol{x}_i$$

$\boldsymbol{x}_i$ is an column vector. Define $\boldsymbol{Z} \in \mathbb{R}^{J \times D} = \{z_{ji}\}_{j=1,i=1}^{J,D}$ and $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{D}$. Then we write above equation as

$$\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{X} \tag{24}$$

The gradient is computed as:

$$\frac{\partial}{\partial \boldsymbol{W}} L(\boldsymbol{W}; \mathcal{D}) = \boldsymbol{H}\boldsymbol{X}^T \tag{25}$$

where $\boldsymbol{H} = (\text{softmax}(\boldsymbol{Z}) - \boldsymbol{Y})$, and we apply softmax function element-wise.

Clearly, we can apply the coded matrix multiplication schemes to computations in (24) and (25). In (24), we encode $\boldsymbol{W}$ and $\boldsymbol{X}$ and perform coded matrix multiplication, then we encode $\boldsymbol{H}$ and reuse encoded $\boldsymbol{X}$ to perform another coded matrix multiplication in (25).

B. Experimental Results

The goal is to explore whether, despite the loss of precision due to approximation, our approach leads to accurate training. We trained the logistic regression using the MNIST dataset [45]. A learning rate of 0.001 and batch size of 128 were used. Each logistic regression experiment was run for 40,000 iterations. For every matrix multiplication step in the training algorithm i.e., computing $\boldsymbol{W}\boldsymbol{X}$ and $\boldsymbol{H}\boldsymbol{X}^T$, we assume that we have N_{succ} successful nodes out of P nodes. Tables I and II show the 10-fold cross validation accuracy results obtained for training and test datasets. For accurate comparison, we used the same the random folds, initialization and batches for the different coding schemes.

We first ran the training algorithm for the worst-case failure scenario where we assume that the worst-case failure pattern happens at every multiplication step. The results are summarized in Table I. To simulate a scenario where different nodes fail in different iterations, we ran the experiments for a scenario where a random subset of N_{succ} nodes returns at every iteration. The corresponding accuracies are given in Table II. The training and test accuracies for uncoded strategy (without failed nodes) are 92.32±0.07 and 91.17±0.25. The training and test accuracies for uncoded distributed strategy (with 2 failed nodes, but no error correction) are 21.45±1.00 and 21.48±1.05 for $m = 5$, and 29.09±5.99 and 28.96±6.32 for $m = 20$. This indicates the importance of redundancy and error correction for accurate model training in presence of stragglers.

From the results in both tables, we observe that approx MatDot codes have essentially identical performance to uncoded multiplication for smaller values of parameter m, but degrades for larger values of m. As expected, the random failure scenario in Table II has much better performance than the worst-case failure scenario. The source code for logistic regression implementation via uncoded and coded multiplications can be found in [44].

V. Discussion and Future Work

This paper opens new directions for coded computing by showing the power of approximations. Specifically, an open research direction is the investigation of related coded computing frameworks (e.g., polynomial evaluations) to examine the gap between ϵ-error and 0-error recovery thresholds.

As our constructions require evaluation points close to 0 (Section III), encoding matrices become ill-conditioned rapidly as m grows. An open direction of future work is to explore numerically stable coding schemes possibly building on recent works, e.g. [39]–[41], with focus on ϵ-error instead of exact computation.

References

[1] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2413–2417.

[2] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," in *Advances In Neural Information Processing Systems (NIPS)*, 2017.

[3] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2020.

[4] 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*, 2016, pp. 2100–2108.

[5] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[6] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 3368–3376.

[7] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic mds codes and expander graphs," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4302–4310.

[8] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2027–2031.

[9] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Information Theory (ISIT), 2017 IEEE International Symposium on*. IEEE, 2017.

[10] H. Jeong, T. M. Low, and P. Grover, "Masterless Coded Computing: A Fully-Distributed Coded FFT Algorithm," in *IEEE Communication, Control, and Computing (Allerton)*, 2018, pp. 887–894.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded fourier transform," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2017, pp. 494–501.

[12] M. Aliasgari, J. Kliewer, and O. Simeone, "Coded computation against processing delays for virtualized cloud-based channel decoding," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 28–38, 2019.

[13] N. S. Ferdinand and S. C. Draper, "Anytime coding for distributed computation," in *Communication, Control, and Computing (Allerton)*, 2016, pp. 954–960.

	Training Accuracy (%)		Test Accuracy (%)	
(m, N_{succ}, P)	Approx MatDot	Chebyshev	Approx MatDot	Chebyshev
(5,5,7)	92.32±0.07	29.00±2.84	91.17±0.26	29.14±2.99
(5,6,8)	92.33±0.07	41.83±4.90	91.16±0.25	41.81±5.38
(5,7,9)	92.33±0.07	50.55±8.00	91.17±0.25	50.44±8.22
(5, 8,10)	92.32±0.07	47.10±5.67	91.17±0.25	46.82±5.63
(5, 9,11)	92.32±0.07	92.32±0.07	91.17±0.25	91.17±0.25
(20,20,22)	44.25±2.06	38.50±8.26	44.04±1.38	38.33±8.16

Table I: Logistic regression results on MNIST dataset for worst case failures

	Training Accuracy (%)		Test Accuracy (%)	
(m, N_{succ}, P)	Approx MatDot	Chebyshev	Approx MatDot	Chebyshev
(5,5,7)	92.32±0.07	91.37±0.10	91.17±0.25	91.03±0.34
(5,6,8)	92.32±0.07	91.91±0.07	91.17±0.25	91.47±0.25
(5,7,9)	92.32±0.07	92.14±0.10	91.17±0.25	91.58±0.16
(5, 8,10)	92.32±0.07	92.22±0.09	91.17±0.25	91.45±0.23
(5, 9,11)	92.32±0.07	92.32±0.07	91.17±0.25	91.17±0.25
(20,20,22)	46.05±1.47	92.54±0.05	45.97±1.02	91.85±0.19

Table II: Logistic regression results on MNIST dataset for random failures

[14] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1620–1624.

[15] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8192–8196.

[16] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5139–5147.

[17] Q. M. Nguyen, H. Jeong, and P. Grover, "Coded QR Decomposition," in *IEEE International Symposium on Information Theory (ISIT)*, 2020.

[18] A. Severinson, A. G. i Amat, and E. Rosnes, "Block-Diagonal and LT Codes for Distributed Computing With Straggling Servers," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1739–1753, 2019.

[19] F. Haddadpour, Y. Yang, V. Cadambe, and P. Grover, "Cross-Iteration Coded Computing," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 196–203.

[20] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 298–304.

[21] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran, "Oversketched newton: Fast convex optimization for serverless systems," *arXiv preprint arXiv:1903.08857*, 2019.

[22] V. Cadambe and P. Grover, "Codes for distributed computing: A tutorial," *IEEE Information Theory Society Newsletter*, vol. 67, no. 4, pp. 3–15, Dec. 2017.

[23] S. Li and S. Avestimehr, *Coded Computing: Mitigating Fundamental Bottlenecks in Large-scale Distributed Computing and Machine Learning*. Now Foundations and Trends, 2020.

[24] S. Dutta, H. Jeong, Y. Yang, V. Cadambe, T. M. Low, and P. Grover, "Addressing Unreliability in Emerging Devices and Non-von Neumann Architectures Using Coded Computing," *Proceedings of the IEEE*, 2020.

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

[26] A. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2809–2816, 2006.

[27] S. B. Balaji, M. N. Krishnan, M. Vajha, V. Ramkumar, B. Sasidharan, and P. V. Kumar, "Erasure coding for distributed storage: an overview," *Science China Information Sciences*, vol. 61, no. 10, p. 100301, 2018. [Online]. Available: https://doi.org/10.1007/s11432-018-9482-6

[28] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[29] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[30] S. Dutta, Z. Bai, H. Jeong, T. Meng Low, and P. Grover, "A Unified Coded Deep Neural Network Training Strategy Based on Generalized PolyDot Codes for Matrix Multiplication," *arXiv preprint arXiv:1811.10751*, 2018.

[31] S. Wang, J. Liu, and N. Shroff, "Fundamental limits of approximate gradient coding," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–22, 2019.

[32] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[33] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic Gradient Coding for Straggler Mitigation in Distributed Learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, 5 2020.

[34] T. Jahani-Nezhad and M. A. Maddah-Ali, "Codedsketch: Coded distributed computation of approximated matrix multiplication," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2489–2493.

[35] ——, "Berrut Approximated Coded Computing: Straggler Resistance Beyond Polynomial Computing," *arXiv preprint arXiv:2009.08327*, 2020.

[36] M. Soleymani, H. Mahdavifar, and A. S. Avestimehr, "Analog lagrange coded computing," *Arxiv preprint, arxiv:2008.08565*, 2020.

[37] J. Kosaian, K. V. Rashmi, and S. Venkataraman, "Parity models: erasure-coded resilience for prediction serving systems," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 30–46.

[38] ——, "Learning-Based Coded Computation," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[39] M. Fahim and V. R. Cadambe, "Numerically Stable Polynomially Coded Computing," *IEEE Transactions on Information Theory*, p. 1, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9319171

[40] A. Ramamoorthy and L. Tang, "Numerically stable coded matrix computations via circulant and rotation matrix embeddings," *Arxiv preprint, arxiv:1910.06515*, 2019.

[41] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random khatri-rao-product codes for numerically-stable distributed matrix multiplication," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 253–259.

[42] N. Charalambides, H. Mahdavifar, and A. O. Hero, "Numerically stable binary gradient coding," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 2622–2627.

[43] L. N. Trefethen, *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.

[44] Github code repository. [Online]. Available: https://github.com/Ateet-dev/ApproxCodedMatrixMulArxiv.git

[45] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[46] E. Cornelius Jr, "Identities for complete homogeneous symmetric polynomials," *JP J. Algebra Number Theory Appl*, vol. 21, no. 1, pp. 109–116, 2011.

APPENDIX A
PROOF OF THEOREM 2

We first prove the following crucial theorem.

Theorem 4. *Let $f(x) = a_0 + a_1 x + \cdot + a_{k-1}x^{k-1}$ and let $x_1, \ldots, x_m$ be distinct real numbers that satisfy $|x_i| \leq \delta$ for all $i = 1, \ldots, m$, for some $0 < \delta < \frac{1}{m}$. Let*

$$\begin{bmatrix} \widehat{a}_0 \\ \vdots \\ \widehat{a}_{m-1} \end{bmatrix} \triangleq \mathbf{V}^{-1} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix}, \tag{26}$$

where $\mathbf{V} = \mathsf{Vander}(\mathbf{x}, m)$ for $\mathbf{x} = [x_1 \ \cdots \ x_m]$. Then,

$$|\widehat{a}_{m-1} - a_{m-1}| \leq ||\mathbf{a}||_\infty \cdot (k-m)m\delta. \tag{27}$$

Proof. Let $R_m(x)$ be the higher order terms in f: $R_m(x) = a_m x^m + \cdots a_{k-1}x^{k-1}$. Then, the following relation holds:

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} = \mathbf{V} \begin{bmatrix} a_0 \\ \vdots \\ a_{m-1} \end{bmatrix} + \begin{bmatrix} R_m(x_1) \\ \vdots \\ R_m(x_m) \end{bmatrix}$$

$$\iff \quad \mathbf{V}^{-1} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} = \begin{bmatrix} a_0 \\ \vdots \\ a_{m-1} \end{bmatrix} + \mathbf{V}^{-1} \begin{bmatrix} R_m(x_1) \\ \vdots \\ R_m(x_m) \end{bmatrix}.$$

Let $\mathbf{v}$ be the last row of $\mathbf{V}^{-1}$, i.e., $\mathbf{v} = \mathbf{V}^{-1}[m, :]$ and let $\mathbf{r} = [R_m(x_i)]_{i\in[m]}$. Then, $|\widehat{a}_{m-1} - a_{m-1}| = |\mathbf{v}\cdot\mathbf{r}|$. Using the explicit formula for the inverse of Vandermonde matrices, the i-th entry of $\mathbf{v}$ is given as:

$$v[i] = \frac{1}{\prod_{\substack{j=1 \\ j\neq i}}^{m}(x_j - x_i)}. \tag{28}$$

Thus, $\mathbf{v}\cdot\mathbf{r}$ can be rewritten as:

$$\begin{aligned} \mathbf{v}\cdot\mathbf{r} &= \sum_{i=1}^{m} \frac{\sum_{l=1}^{k-m} a_{m-1+l} x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j\neq i}}^{m}(x_j - x_i)} \\ &= \sum_{l=1}^{k-m} a_{m-1+l} \sum_{i=1}^{m} \frac{x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j\neq i}}^{m}(x_j - x_i)}. \end{aligned} \tag{29}$$

The expression in (29) can be further simplified using the following lemma.

Lemma 1. *[Theorem 3.2 in [46]]*

$$\sum_{i=1}^{m} \frac{x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j\neq i}}^{m}(x_j - x_i)} = h_l(x_1, \ldots, x_m), \tag{30}$$

where h_l is the complete homogeneous symmetric polynomial of degree l defined as:

$$h_l(x_1, \ldots, x_m) = \sum_{d_1 + \cdots + d_m = l} x_1^{d_1} \cdot x_2^{d_2} \cdots x_m^{d_m}. \tag{31}$$

Using Lemma 1, (29) can now be written as: $\mathbf{v}\cdot\mathbf{r} = \sum_{l=1}^{k-m} a_{m-1+l} \cdot h_l(x_1, \cdots, x_m)$. Finally,

$$\begin{aligned} |\widehat{a}_{m-1} - a_{m-1}| &= |\mathbf{v}\cdot\mathbf{r}| \\ &= \left| \sum_{l=1}^{k-m} a_{m-1+l} \cdot h_l(x_1, \cdots, x_m) \right| \\ &\leq \sum_{l=1}^{k-m} a_{m-1+l} \cdot \binom{m+l-1}{l} \delta^l \\ &\leq ||\mathbf{a}||_\infty \sum_{l=1}^{k-m} \binom{m+l-1}{l} \delta^l \\ &\leq ||\mathbf{a}||_\infty (k-m)m\delta. \end{aligned} \tag{32}$$

The last inequality holds because $\delta < \frac{1}{m}$ and thus $\binom{m+l-1}{l}\delta^l \leq \binom{m}{1}\delta$ for $l = 1, \ldots, m-1$. □

Recall that polynomial $p_{\mathbf{C}}(x)$ is essentially a set of n^2 polynomials, having one polynomial for each $C[i,j]$ $(i,j \in [n]$, and we use $p_{C[i,j]}(x)$ as the (i,j)-th polynomial for $C[i,j]$.

Lemma 2. *Assume $||\mathbf{A}||_F \leq \eta$ and $||\mathbf{B}||_F \leq \eta$. Then, for $p_{\mathbf{C}}(x)$ given in Construction 2, the ∞-norm of $\mathbf{p}_{[v,w]} = \mathsf{vec}(p_{C[v,w]})$ $(v,w \in [n])$ is bounded as:*

$$||\mathbf{p}_{[v,w]}||_\infty \leq \eta^2. \tag{33}$$

Proof. Let $d \triangleq q(i-1) + pq(l-1) + (q-1+j-k)$. The coefficient of y^d in $p_{\mathbf{C}}(x)$ is:

$$\mathbf{P}_d = \begin{cases} \sum_{j'-k'=j-k} \mathbf{A}_{i,j'}\mathbf{B}_{k',l} \\ \quad + \sum_{j'-k'=j-k-q} \mathbf{A}_{i+1,j'}\mathbf{B}_{k',l}, & \text{for } j-k>0, \\ \sum_{j'=k'} \mathbf{A}_{i,j'}\mathbf{B}_{k',l}, & \text{for } j-k=0. \end{cases} \tag{34}$$

For both cases, the number of terms in the sum is q. Thus, it can be rewritten as:

$$\begin{bmatrix}\mathbf{A}_{j_1} & \cdots & \mathbf{A}_{j_q}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{B}_{k_1} \\ \vdots \\ \mathbf{B}_{k_q}\end{bmatrix}. \tag{35}$$

As these matrices are submatrices of $\mathbf{A}$ and $\mathbf{B}$,

$$\left\|\begin{bmatrix}\mathbf{A}_{j_1} & \cdots & \mathbf{A}_{j_q}\end{bmatrix}\right\|_2 \leq \eta, \quad \left\|\begin{bmatrix}\mathbf{B}_{k_1} \\ \vdots \\ \mathbf{B}_{k_q}\end{bmatrix}\right\|_2 \leq \eta, \tag{36}$$

Hence, $||\mathbf{p}_{[v,w]}||_\infty = \max_d |P_d[v,w]| \leq ||\mathbf{P}_d||_2 \leq \eta^2$. □

Proof of Theorem 2. The decoding for ϵ-approximate PolyDot codes can be performed as follows. For decoding $\mathbf{C}_{i,l}$, we choose $d_{i,l} = iq + pq(l-1)$ points from the p^2q successful nodes. Let $\mathbf{V}_{i,l} = \mathsf{Vander}([x_1, \cdots, x_{d_{i,l}}], d_{i,l})$ and $\mathbf{v}$ be the last row of $\mathbf{V}_{i,l}^{-1}$. Then, we decode $\mathbf{C}_{i,l}$ by computing:

$$\widehat{\mathbf{C}}_{i,l} = \mathbf{v} \cdot \begin{bmatrix} p_{\mathbf{C}}(x_1) \\ \vdots \\ p_{\mathbf{C}}(x_{d_{i,l}}) \end{bmatrix}. \tag{37}$$

By combining Theorem 4 and Lemma 2, we can show that:

$$\left\|\widehat{\mathbf{C}}_{i,l} - \mathbf{C}_{i,l}\right\|_{\max} \leq \frac{(p^2q+q-1-d_{i,l})d_{i,l}}{q(p^2q-1)}\epsilon.$$

The smallest $d_{i,l}$ is $d_{1,1} = q$ and the largest $d_{i,l}$ is $d_{p,p} = p^2q$. For $q \leq d_{i,l} \leq p^2q$, $(p^2q+q-1-d_{i,l})d_{i,l} \leq q(p^2q-1)$. Hence, $\left\|\widehat{\mathbf{C}}_{i,l} - \mathbf{C}_{i,l}\right\|_{\max} \leq \epsilon$. □

APPENDIX B

Proof of Theorem 3. We provide proof by contradiction. Assume that there exists a coding scheme with $K(p,q,\epsilon) = pq-1$, $\forall \epsilon < \epsilon_0$, for some constant ϵ_0, i.e., linear encoding functions f_i's, g_i's and decoding function $d_{\mathcal{S}}$ such that $||d_{\mathcal{S}}((\widetilde{\mathbf{C}}_i)_{i\in\mathcal{S}}) - \mathbf{AB}||_{\max} \leq \epsilon$ for all $|\mathcal{S}| \geq pq-1$.

Since f_i is a linear encoding function, it can be written as:

$$f_i(\mathbf{A}_{1,1}, \ldots, \mathbf{A}_{p,q}) = \gamma_{i,1}\mathbf{A}_{1,1} + \gamma_{i,2}\mathbf{A}_{1,2} + \cdots + \gamma_{i,pq}\mathbf{A}_{p,q}$$

Now, let $\mathcal{S} \subset [P]$ such that $|\mathcal{S}| = pq-1$ and let $\mathbf{\Gamma}$ be a $(pq-1) \times pq$ matrix: $\mathbf{\Gamma} = [\gamma_{i,j}]_{i\in\mathcal{S}, j\in[pq]}$. Let $\boldsymbol{\rho}$ be a unit null vector of $\mathbf{\Gamma}$. We reshape $\boldsymbol{\rho}$ in to a $p \times q$ matrix $\boldsymbol{Q}$ in a row-major order. Then, we let

$$\overline{\mathbf{A}} = \frac{\eta\sqrt{pq}}{n}\mathbf{1}_{\frac{n}{p}\times\frac{n}{q}}, \quad \overline{\mathbf{B}} = \frac{\eta\sqrt{pq}}{n}\mathbf{1}_{\frac{n}{q}\times\frac{n}{p}},$$

and let $\mathbf{A} = \boldsymbol{Q} \otimes \overline{\mathbf{A}}, \mathbf{B} = \boldsymbol{Q}^T \otimes \overline{\mathbf{B}}$, where $\mathbf{1}_{\frac{n}{p}\times\frac{n}{q}}$ and $\mathbf{1}_{\frac{n}{q}\times\frac{n}{p}}$ are all-one matrices of dimension $\frac{n}{p}\times\frac{n}{q}$ and $\frac{n}{q}\times\frac{n}{p}$. Since $\boldsymbol{Q}$ is constructed from the null space of $\mathbf{\Gamma}$, notice that for all $i \in \mathcal{S}$, $f_i(\mathbf{A}) = \mathbf{0}$. Hence, for all $i \in \mathcal{S}$, the output at the i-th node $\widetilde{\mathbf{C}}_i = f_i(\mathbf{A})g_i(\mathbf{B}) = \mathbf{0}$. This implies that

$$d_{\mathcal{S}}((\widetilde{\boldsymbol{C}}_i)_{i\in\mathcal{S}}) = d_{\mathcal{S}}((-\widetilde{\boldsymbol{C}}_i)_{i\in\mathcal{S}}) \overset{\text{define}}{=} d_{\mathcal{S}}(\mathbf{0}).$$

By triangle inequality,

$$\begin{aligned} &||d_{\mathcal{S}}(\widetilde{\boldsymbol{C}}) - \mathbf{AB}||_F + ||d_{\mathcal{S}}(-\widetilde{\boldsymbol{C}}) - \mathbf{AB}||_F \\ &\quad \geq ||\mathbf{AB} - (-\mathbf{AB})||_F = 2||\mathbf{AB}||_F. \end{aligned}$$

Hence,

$$\max(||d_{\mathcal{S}}(\mathbf{0}) - \mathbf{AB}||_F, ||d_{\mathcal{S}}(\mathbf{0}) + \mathbf{AB}||_F) \geq ||\mathbf{AB}||_F.$$

Assume $\mathrm{rank}(\boldsymbol{Q}) = r$.

$$\begin{aligned} ||\mathbf{AB}||_F &= ||(\boldsymbol{Q}\otimes\overline{\mathbf{A}})(\boldsymbol{Q}^T\otimes\overline{\mathbf{B}})||_F \\ &= ||(\boldsymbol{Q}\boldsymbol{Q}^T)\otimes(\overline{\mathbf{AB}})||_F \\ &= \sqrt{\mathrm{Tr}((\boldsymbol{Q}\boldsymbol{Q}^T\otimes\overline{\mathbf{B}}^T\overline{\mathbf{A}}^T)(\boldsymbol{Q}\boldsymbol{Q}^T\otimes\overline{\mathbf{AB}}))} \\ &= \sqrt{\mathrm{Tr}((\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{Q}^T)\otimes(\overline{\mathbf{B}}^T\overline{\mathbf{A}}^T\overline{\mathbf{AB}}))} \\ &= \sqrt{\mathrm{Tr}(\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{Q}^T)}\sqrt{\mathrm{Tr}(\overline{\mathbf{B}}^T\overline{\mathbf{A}}^T\overline{\mathbf{AB}})} \\ &= ||\boldsymbol{Q}\boldsymbol{Q}^T||_F||\overline{\mathbf{AB}}||_F \\ &\overset{(a)}{\geq} ||\boldsymbol{Q}\boldsymbol{Q}^T||_2||\overline{\mathbf{AB}}||_F = ||\boldsymbol{Q}||_2^2||\overline{\mathbf{AB}}||_F \\ &\overset{(b)}{\geq} \frac{||\boldsymbol{Q}||_F^2}{r}||\overline{\mathbf{AB}}||_F \overset{(c)}{\geq} \frac{||\boldsymbol{Q}||_F^2}{\min(p,q)}||\overline{\mathbf{AB}}||_F \\ &= \frac{\eta^2}{\min(p,q)} \end{aligned}$$

The following inequalities were used: $r \overset{(c)}{\leq} \min(p,q)$, and for some matrix $\mathbf{Z}$, $||\mathbf{Z}||_2 \overset{(a)}{\leq} ||\mathbf{Z}||_F \overset{(b)}{\leq} \sqrt{r}||\mathbf{Z}||_2$. Therefore

$$\begin{aligned}\max(||d_{\mathcal{S}}(\mathbf{0}) - \mathbf{AB}||_F, ||d_{\mathcal{S}}(\mathbf{0}) + \mathbf{AB}||_F) &\geq \frac{\eta^2}{\min(p,q)} \\ \implies \max(||d_{\mathcal{S}}(\mathbf{0}) - \mathbf{AB}||_{\max}, ||d_{\mathcal{S}}(\mathbf{0}) + \mathbf{AB}||_{\max}) &\\ \geq \frac{\eta^2}{n \cdot \min(p,q)} \triangleq \epsilon_0 &\end{aligned}$$

Therefore, for at least one of $\mathbf{AB}$ and $-\mathbf{AB}$, the decoding error is $\geq \frac{\eta^2}{n \cdot \min(p,q)}$ when recovery threshold is set to $pq - 1$.

□