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

Viveck R. Cadambe[†*], Flavio P. Calmon[‡*], Ateet Devulapalli[†*], Haewon Jeong[‡*]
[†]Penn State University, [‡] Harvard University

***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 code previously developed in the context of exact recovery ($\epsilon = 0$). Since previous results showed that the MatDot code is tight for a class of linear coding schemes for exact recovery, our result shows that allowing for mild approximations leads to a system that is nearly twice as efficient as exact reconstruction. Moreover, we develop an optimization framework based on alternating minimization that enables the discovery of new codes for approximate matrix multiplication.**

Theorem proofs and other missing details are provided in the extended version of the paper [1].

I. Introduction

Coded computing has emerged as a promising paradigm to resolving straggler and security bottlenecks in large-scale distributed computing platforms [2]–[25]. 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 show 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 [4]. 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] [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 [6] 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, i.e., elementary linear transformations that were originally studied in [28]. While the results of [4], [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 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 [3], [4]; 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 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 [4], [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 the our result is tight for $0 < \epsilon < 1$ for unit norm matrices. 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 believe that these results open up a new avenue in coded computing research via revisiting existing code constructions and allowing for an ϵ-error.

A second contribution of our paper is the development of an optimization formulation that enables the discovery of new coding schemes for approximate computing. We show that the optimization can be solved through an alternating minimization algorithm that has simple, closed-form iterations as well as provable convergence to a local minimum. We demonstrate through numerical examples that our optimization approach

*Authors in alphabetical order. This material is based upon work supported by the National Science Foundation under grants CIF 1900750, CAREER 1845852, and CCF 1763657.

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

finds approximate computing codes with favourable trade-offs between approximation error and recovery threshold. Finally, we demonstrate that the proposed approximate code computing strategies can be used in practical machine learning algorithms through an example application of logistic regression.

B. Related Work

The study of coded computing for elementary linear algebra operations, starting from [5], [28], is an active research (see surveys [23]–[25]). Notably, the recovery thresholds for matrix multiplication were established via achievability and converse results respectively in [3], [4], [29]. The Lagrange coded computing framework of [6] generalized the systematic MatDot code construction of [4] 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 [30], [31] studied the idea of gradient coding from an approximation viewpoint. The references that are most relevant to our work are [21], [22], [32]. Like us, these references 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 [32] is particularly similar to ours. Our results can be viewed as strict improvement, 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 these references. A related line of work in [33], [34] study coded polynomial evaluation beyond exact recovery and note techniques to improve the quality of the approximation. Yet, we are the first to establish the strict gap in the recovery thresholds for ϵ-error computations versus exact computation for matrix multiplication.

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

$$||\mathbf{A}||_F \leq \eta \quad \text{and} \quad ||\mathbf{B}||_F \leq \eta, \tag{1}$$

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 break down $\mathbf{A}, \mathbf{B}$ into small equal-sized sub-blocks as follows[2]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,q} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{p,1} & \cdots & \mathbf{A}_{p,q} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,p} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{q,1} & \cdots & \mathbf{B}_{q,p} \end{bmatrix}, \tag{2}$$

where $pq = m$.

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:

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

where

$$f_i : \underbrace{\mathbb{R}^{\frac{n}{p} \times \frac{n}{q}} \times \cdots \times \mathbb{R}^{\frac{n}{p} \times \frac{n}{q}}}_{pq=m} \to \mathbb{R}^{\frac{n}{p} \times \frac{n}{q}}, \tag{3}$$

$$g_i : \underbrace{\mathbb{R}^{\frac{n}{q} \times \frac{n}{p}} \times \cdots \times \mathbb{R}^{\frac{n}{q} \times \frac{n}{p}}}_{m} \to \mathbb{R}^{\frac{n}{q} \times \frac{n}{p}}. \tag{4}$$

We assume that f_i, g_i are linear, i.e., their outputs are linear combinations of m inputs. For example, we may have

$$f_i(\mathbf{V}_1, \cdots, \mathbf{V}_m) = \gamma_{i,1}\mathbf{V}_1 + \cdots + \gamma_{i,m}\mathbf{V}_m \tag{5}$$

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

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

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 encoding functions:

$$\mathbf{f} = \begin{bmatrix} f_1 & \cdots & f_P \end{bmatrix}, \mathbf{g} = \begin{bmatrix} g_1 & \cdots & g_P \end{bmatrix}, \tag{8}$$

and let $\mathbf{d}$ be a length-2^P vector of decoding functions $d_{\mathcal{S}}$ for all subsets $\mathcal{S} \subseteq [P]$. Then, we say that the ϵ-approximate recovery threshold of $\mathbf{f}, \mathbf{g}, \mathbf{d}$ is K if

$$|\widehat{C}_{\mathcal{S}}[i, j] - C[i, j]| \leq \epsilon \quad (i, j \in [n]), \tag{9}$$

for every $\mathcal{S} \subseteq [P]$ such that $|\mathcal{S}| \geq K$, and any $\mathbf{A}$ and $\mathbf{B}$ that satisfy the norm constraints (1). We denote this recovery threshold as $K(m, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d})$. Moreover, let $K^*(m, \epsilon)$ be defined as the minimum of $K(m, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d})$ over all possible linear functions $\mathbf{f}$, $\mathbf{g}$ and decoding functions $\mathbf{d}$, i.e.,

$$K^*(m, \epsilon) \triangleq \min_{\mathbf{f}, \mathbf{g}, \mathbf{d}} K(m, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d}). \tag{10}$$

[2]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.

Note that parameters p and q are embedded in $\mathbf{f}$ and $\mathbf{g}$. Through an achievability scheme in [4] 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 [4])**.** *Under the system model given in Section II-B*

$$K^*(m, \epsilon = 0) = 2m - 1. \tag{11}$$

Our main result is the following:

Theorem 2. *Under the system model given in Section II-B, the optimal ϵ-approximate recovery threshold is:*

$$K^*(m, \epsilon) = m. \tag{12}$$

(Achievability – Theorem 3)
For any $0 < \epsilon < \min(2, 3\eta^2\sqrt{2m-1})$, the ϵ-approximate MatDot codes in Construction 2 achieves:

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

(Converse – Theorem 4)
For all $0 < \epsilon < \eta^2$, $K^*(m, \epsilon) \geq m$.

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

In this section, we first propose the construction of ϵ-Approximate MatDot codes that can achieve the recovery threshold of m for ϵ approximation error. Then, we show the converse result which shows that the recovery threshold cannot be smaller than m for sufficiently small ϵ.

A. Approximate MatDot Codes

We briefly introduce the construction of MatDot codes and then we show that a simple adaptation of MatDot codes can be used for approximate coded computing.

Construction 1 (MatDot Codes [4])**.** *Define polynomials $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ as follows:*

$$p_{\mathbf{A}}(x) = \sum_{i=1}^{m} \mathbf{A}_i x^{i-1}, p_{\mathbf{B}}(x) = \sum_{j=1}^{m} \mathbf{B}_j x^{m-j}. \tag{13}$$

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:

$$p_{\mathbf{C}}(x) = \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbf{A}_i \mathbf{B}_j x^{m-1+(i-j)}. \tag{14}$$

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 output 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 $p_{\mathbf{C}}(x)$. The key idea of Approximate MatDot Codes is to choose the evaluation points carefully to reduce this overhead. In fact, we have to choose evaluation points in a small interval that is proportional to ϵ.

Construction 2 (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 such that*

$$0 < \epsilon < \min(2, 3\eta^2\sqrt{2m-1}). \tag{15}$$

Then, ϵ-Approximate MatDot code is a MatDot code defined in Construction 1 with evaluation points $\lambda_1, \ldots, \lambda_P$ that satisfy:

$$|\lambda_i| < \frac{\epsilon}{6\eta^2\sqrt{2m-1}(m-1)m}, \quad i \in [P]. \tag{16}$$

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

Theorem 3. *For any $0 < \epsilon < \min(2, 3\eta^2\sqrt{2m-1})$, the ϵ-approximate MatDot codes in Construction 2 achieves:*

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

where $\mathbf{f}_{\epsilon\text{-MatDot}}, \mathbf{g}_{\epsilon\text{-MatDot}}, \mathbf{d}_{\epsilon\text{-MatDot}}$ are encoding and decoding functions specified by Construction 2.

Remark 1. *The error bound provided by Theorem 3 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, the upper bound of $||\mathbf{A}||_F$ and $||\mathbf{B}||_F$ as given in* (16)*. 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.*

While we defer the full proof to [1], we provide an intuitive explanation of the above theorem.

B. An insight behind Approximate MatDot Codes

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:

$$S(x) = P(x)Q(x) + R(x), \tag{17}$$

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,

$$S(\lambda_i) = R(\lambda_i). \tag{18}$$

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:

$$S(x) = x^m Q(x) + R(x). \tag{19}$$

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:

$$S(\lambda_i) = \lambda_i^m Q(\lambda_i) + R(\lambda_i) \approx R(\lambda_i). \tag{20}$$

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 (1) on the input matrices and, thus, Q must have a bounded norm since the higher-order terms in S are solely determined by Q.

C. Converse

We have shown that for *any* matrices $\mathbf{A}$ and $\mathbf{B}$, and with a recovery threshold of m, MatDot codes can achieve arbitrarily small error. We now show a converse indicating that for a recovery threshold of $m-1$, there exists matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$ where the error cannot be made arbitrarily small for any type of encoding.

Theorem 4. *Under the system model given in Section II, for any* $0 < \epsilon < \eta^2$,

$$K^*(m, \epsilon) \geq m. \tag{21}$$

IV. An Optimization Approach to Approximate Coded Computing and Its Application in Machine Learning

The construction of Approximate MatDot code shows the theoretical possibility that the approximate recovery threshold can be brought down to m from $2m-1$. In this section, we propose another, optimization-based approach to find an approximate coded computing scheme. The goal of optimization is to find ϵ such that $K^*(m, \epsilon) \leq k$ for a given k, among linear encoding and decoding functions. In Section IV-A, we illustrate our optimization framework through a simple example, and in Section IV-B, we state the framework in its full generality. W e show the results of our optimization algorithm in Section IV-C.

A. A simple example ($m = 2, k = 2, P = 3$)

The input matrices are split into:

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix}\mathbf{B}_1 \\ \mathbf{B}_2\end{bmatrix}.$$

As f_i's and g_i's are linear encoding functions, let

$$\boldsymbol{\alpha}^{(i)} = \begin{bmatrix}\alpha_1^{(i)} \\ \alpha_2^{(i)}\end{bmatrix}, \boldsymbol{\beta}^{(i)} = \begin{bmatrix}\beta_1^{(i)} \\ \beta_2^{(i)}\end{bmatrix}$$

be the encoding coefficients for $\mathbf{A}$ and $\mathbf{B}$ for the i-th node. The i-th worker node receives encoded inputs:

$$\widetilde{\mathbf{A}}^{(i)} = \alpha_1^{(i)}\mathbf{A}_1 + \alpha_2^{(i)}\mathbf{A}_2, \;\; \widetilde{\mathbf{B}}^{(i)} = \beta_1^{(i)}\mathbf{B}_1 + \beta_2^{(i)}\mathbf{B}_2.$$

The matrix product output at the i-th worker node is:

$$\begin{aligned}\widetilde{\mathbf{C}}^{(i)} = \widetilde{\mathbf{A}}^{(i)}\widetilde{\mathbf{B}}^{(i)} &= \alpha_1^{(i)}\beta_1^{(i)} \cdot \mathbf{A}_1\mathbf{B}_1 + \alpha_1^{(i)}\beta_2^{(i)} \cdot \mathbf{A}_1\mathbf{B}_2 \\ &+ \alpha_2^{(i)}\beta_1^{(i)} \cdot \mathbf{A}_2\mathbf{B}_1 + \alpha_2^{(i)}\beta_2^{(i)} \cdot \mathbf{A}_2\mathbf{B}_2.\end{aligned}$$

The recovery threshold $k = 2$ implies that with any two $\widetilde{\mathbf{C}}^{(i)}$, $\widetilde{\mathbf{C}}^{(j)}$, $i \neq j,\ i, j \in [3]$, the master node can recover:

$$\begin{aligned}\mathbf{C} &= \mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_2 \\ &= 1 \cdot \mathbf{A}_1\mathbf{B}_1 + 0 \cdot \mathbf{A}_1\mathbf{B}_2 + 0 \cdot \mathbf{A}_2\mathbf{B}_1 + 1 \cdot \mathbf{A}_2\mathbf{B}_2.\end{aligned}$$

For illustration, assume that nodes $i = 1$ and $j = 2$ responded first. For linear decoding, our goal is to determine decoding coefficients $d_1, d_2 \in \mathbb{R}$ that yield: $\mathbf{C} = d_1\widetilde{\mathbf{C}}^{(1)} + d_2\widetilde{\mathbf{C}}^{(2)}$. For this to hold for any $\mathbf{A}$ and $\mathbf{B}$, the coefficients must satisfy:

$$\begin{aligned}\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix} = d_1 &\begin{bmatrix}\alpha_1^{(1)}\beta_1^{(1)} & \alpha_1^{(1)}\beta_2^{(1)} & \alpha_2^{(1)}\beta_1^{(1)} & \alpha_2^{(1)}\beta_2^{(1)}\end{bmatrix} \\ + d_2 &\begin{bmatrix}\alpha_1^{(2)}\beta_1^{(2)} & \alpha_1^{(2)}\beta_2^{(2)} & \alpha_2^{(2)}\beta_1^{(2)} & \alpha_2^{(2)}\beta_2^{(2)}\end{bmatrix}\end{aligned} \tag{22}$$

By reshaping the length-4 vectors in (22) into 2×2 matrices and denoting the identity matrix by $\mathbf{I}_{2\times 2}$, (22) is equivalent to

$$\mathbf{I}_{2\times 2} = \sum_{i=1}^{2} d_i \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}. \tag{23}$$

Encoding coefficients $\boldsymbol{\alpha}^{(i)}$'s, $\boldsymbol{\beta}^{(i)}$'s and the decoding coefficients d_i's that satisfy the equality in (23) would guarantee exact recovery for any input matrices $\mathbf{A}$ and $\mathbf{B}$. However, we are interested in *approximate* recovery, which means that we want the LHS and RHS in (23) to be approximately equal. Hence, the goal of optimization is to find encoding and decoding coefficients that minimize the difference between LHS and RHS in (23). One possible objective function for this is:

$$||\mathbf{I}_{2\times 2} - \sum_{i=1}^{2} d_i \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}||_F^2. \tag{24}$$

Recall that this is for the scenario where the third node fails and the first two nodes are successful. There are $\binom{3}{2} = 3$ scenarios where two nodes out of three nodes are successful. For the final objective function, we have to add such loss function for each of these three scenarios. We formalize this next.

B. Optimization Formulation

We formulate the optimization framework for arbitrary values of m, k and P. We denote the encoding coefficients for the i-th node as:

$$\boldsymbol{\alpha}^{(i)} = [\alpha_1^{(i)}, \cdots, \alpha_m^{(i)}]^T, \qquad \boldsymbol{\beta}^{(i)} = [\beta_1^{(i)}, \cdots, \beta_m^{(i)}]^T.$$

Let $\mathcal{P}_k([P]) = \{\mathcal{S} : \mathcal{S} \subseteq [P], |\mathcal{S}| = k\}$ and let $\mathcal{S}_p$ be the p-th set in $\mathcal{P}_k([P])$. In other words, $\mathcal{P}_k([P])$ is a set of all failure scenarios with k successful nodes out of P nodes. Then, we define $\mathbf{d}^{(p)}$ as the vector of decoding coefficients when $\mathcal{S}_p$ is the set of successful workers. We define our optimization problem as follows:

Optimization for Approximate Coded Computing:

$$\min_{\substack{\boldsymbol{\alpha}^{(i)},\, \boldsymbol{\beta}^{(i)},\, \mathbf{d}^{(p)} \\ i=1,\ldots,n, \\ p=1,\ldots,\binom{P}{k}}} \sum_{p=1}^{\binom{P}{k}} ||\mathbf{I}_{m\times m} - \sum_{i\in\mathcal{S}_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}||_F^2. \tag{25}$$

Notice that (25) is a non-convex problem, but it is convex with respect to each coordinate, i.e., with respect to $\{\boldsymbol{\alpha}^{(i)} : i \in [n]\}$, $\{\boldsymbol{\beta}^{(i)} : i \in [n]\}$, and $\{\mathbf{d}^{(p)} : p \in \left[\binom{P}{k}\right]\}$. Hence, we

Symbol	Dimension	Expression
$\boldsymbol{\mathcal{A}}$	$m \times P$	$\left[\boldsymbol{\alpha}^{(1)} \quad \cdots \quad \boldsymbol{\alpha}^{(P)}\right]$
$\boldsymbol{\mathcal{B}}$	$m \times P$	$\left[\boldsymbol{\beta}^{(1)} \quad \cdots \quad \boldsymbol{\beta}^{(P)}\right]$
$\mathbf{Z}^{(\text{full})}$	$P \times P$	$(\boldsymbol{\mathcal{A}}^T\boldsymbol{\mathcal{A}}) \odot (\boldsymbol{\mathcal{B}}^T\boldsymbol{\mathcal{B}})$
$\mathbf{z}^{(\text{full})}$	P	$[\boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\beta}^{(i)}]_{i=1,\ldots,P}$
$\mathbf{Z}^{(p)}$	$k \times k$	$\mathbf{Z}^{(\text{full})}\big\|_{i\in\mathcal{S}_p, j\in\mathcal{S}_p}$
$\mathbf{z}^{(p)}$	k	$\mathbf{z}^{(\text{full})}\big\|_{i\in\mathcal{S}_p}$
$\mathbf{Y}$	$P \times P$	$\left[\sum_{p:i,j\in\mathcal{S}_p} d_i^{(p)} d_j^{(p)}\right]_{\substack{i=1,\ldots,P \\ j=1,\ldots,P}}$
$\mathbf{y}$	P	$\left[\sum_{p:i\in\mathcal{S}_p} d_i^{(p)}\right]_{i=1,\ldots,P}$
$\mathbf{Y}_{\boldsymbol{\mathcal{A}}}$, $\mathbf{Y}_{\boldsymbol{\mathcal{B}}}$	$P \times P$	$\mathbf{Y}_{\boldsymbol{\mathcal{A}}} = \mathbf{Y} \odot (\boldsymbol{\mathcal{A}}^T\boldsymbol{\mathcal{A}})$, $\mathbf{Y}_{\boldsymbol{\mathcal{B}}} = \mathbf{Y} \odot (\boldsymbol{\mathcal{B}}^T\boldsymbol{\mathcal{B}})$

TABLE I
SUMMARY OF NOTATIONS USED IN PROPOSITION 1 AND ALGORITHM 1

propose an alternating minimization algorithm that minimizes for $\mathbf{d}^{(p)}$, $\boldsymbol{\alpha}^{(i)}$, and $\boldsymbol{\beta}^{(i)}$ sequentially. Each minimization step is a quadratic optimization with a closed-form solution, which we describe in the following proposition. The notation used in the proposition and in Algorithm 1 is summarized in Table I

Proposition 1. *The stationary points of the objective function given in* (25) *satisfy*

(i) $\mathbf{Z}^{(p)} \cdot \mathbf{d}^{(p)} = \mathbf{z}^{(p)}$ *for* $p = 1, \ldots, \binom{n}{k}$,

(ii) $\mathbf{Y}_{\boldsymbol{\mathcal{B}}}\boldsymbol{\mathcal{A}} = \text{diag}(\mathbf{y})\boldsymbol{\mathcal{B}}$,

(iii) $\mathbf{Y}_{\boldsymbol{\mathcal{A}}}\boldsymbol{\mathcal{B}} = \text{diag}(\mathbf{y})\boldsymbol{\mathcal{A}}$,

where $\text{diag}(\mathbf{y})$ *is an* n*-by-*n *matrix which has* y_i *on the* i*-th diagonal and* 0 *elsewhere.*

Algorithm 1 sequentially solves conditions (i)–(iii) in Proposition 1. Since each step corresponds to minimizing (25) for one of the variables $\mathbf{d}^{(p)}$, $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{\mathcal{B}}$, the resulting objective is non-increasing in the algorithm's iterations and converges to a local minimum.

Algorithm 1: Alternating Quadratic Minimization

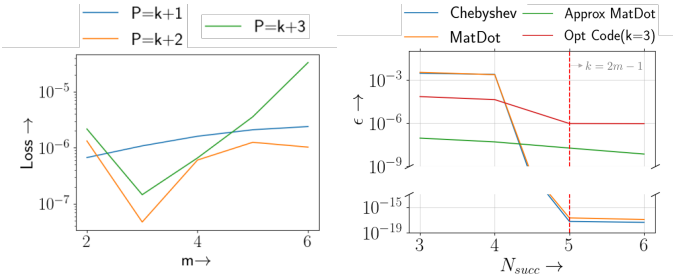
Input: Positive Integers m, k and P $(P > k)$;
Output: $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, $\mathbf{d}^{(p)}$ $(p = 1, \ldots, P)$;
Initialize: Random $m \times P$ matrices $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$;
while *num_iter* < *max_iter* **do**
  Compute $\mathbf{Z}^{(\text{full})}$ and $\mathbf{z}^{(\text{full})}$ from $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$;
  for $p \leftarrow 1$ **to** P **do**
    Solve for $\mathbf{d}^{(p)}$: $\mathbf{Z}^{(p)}\mathbf{d}^{(p)} = \mathbf{z}^{(p)}$
  end
  Compute $\mathbf{Y}$ and $\mathbf{y}$, and $\mathbf{Y}_{\boldsymbol{\mathcal{B}}}$;
  $\boldsymbol{\mathcal{A}} \leftarrow \boldsymbol{\mathcal{A}}^*$, $\boldsymbol{\mathcal{A}}^*$: solution of $\mathbf{Y}_{\mathcal{B}} \cdot \boldsymbol{\mathcal{A}} = \text{diag}(\mathbf{y}) \cdot \boldsymbol{\mathcal{B}}$;
  Compute $\mathbf{Y}_{\boldsymbol{\mathcal{A}}}$;
  $\boldsymbol{\mathcal{B}} \leftarrow \boldsymbol{\mathcal{B}}^*$, $\boldsymbol{\mathcal{B}}^*$: solution of $\mathbf{Y}_{\boldsymbol{\mathcal{A}}} \cdot \boldsymbol{\mathcal{B}} = \text{diag}(\mathbf{y}) \cdot \boldsymbol{\mathcal{A}}$;
end

We next show how the optimization objective in (25) is related to the relative error of the computation output. Let $\ell^{(p)}$ be the loss function for the p-th scenario, i.e.,

$$\ell^{(p)} = ||\mathbf{I}_{m\times m} - \sum_{i\in\mathcal{S}_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)} \boldsymbol{\beta}^{(i)T}||_F^2. \tag{26}$$

Theorem 5. *The error between the decoded result from the nodes in* $\mathcal{S}_p$, $\widehat{\mathbf{C}}_{\mathcal{S}_p}$, *and the true result* $\mathbf{C}$ *can be bounded as:*

$$||\mathbf{C} - \widehat{\mathbf{C}}_{\mathcal{S}_p}||_F \leq \sqrt{\ell^{(p)}} \cdot m \cdot \eta^2. \tag{27}$$



(a) (b)

Fig. 1. (a) Results of running Algorithm 1 with $k = 2m - 2$. The y-axis is min loss from 1,000 different trials. (b) Comparison of exact and approximate coding strategies. The y-axis is the empirical evaluation of ϵ, i.e., $||\widehat{\mathbf{C}} - \mathbf{C}||_{\max}$.

C. Experimental Results

In Fig. 1a, we plot the result of running Algorithm 1 for different (m, k, P) pairs while fixing $k = 2m - 2$, and we take the best code out of 1,000 trials with different random initializations. For each trial, we ran the optimization for 1,000,000 iterations. It demonstrates the best codes found have a loss ($\sim 10^{-5}$) that is much smaller than $1/m^2$, which implies accurate reconstruction due to Theorem 5. The figure thus demonstrates the power of the optimization framework. In Fig. 1b, we compare the performance of exact-recovery codes (conventional MatDot codes and Chebyshev polynomial-based codes [35]) with the approximate MatDot codes and optimization codes developed in this paper. The parameter N_{succ} represents the number of non-straggling nodes. The figure demonstrates that for $N_{\text{succ}} \geq k = 2m - 1$, the Matdot and Chebyshev codes have very small loss and error (10^{-16}), however, for $N_{\text{succ}} < k$ Approximate MatDot codes and optimized codes outperform Chebyshev and MatDot codes.

D. Application to Logistic Regression

Finally, we show that the proposed approximate coded computing strategies can be used in machine learning applications with little impact on accuracy. We trained a logistic regression model for the MNIST dataset where we applied Approximate MatDot codes and optimized codes to every matrix multiplication step in the training, assuming that we only get k successful outputs from P nodes. The test accuracy for uncoded strategy (without failed nodes) was 91.17 ± 0.25. Even when $k < 2m - 1$, e.g., $(m, k, P) = (5, 5, 7)$, training with Approximate MatDot codes achieves nearly the same accuracy as uncoded performance, i.e., 91.17 ± 0.26. More extensive experimental results and details can be found in [1].

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. Since our constructions requires evaluation points close to 0 (Section III), we potentially get ill-conditioned encoding matrices. An open direction of future work is to explore numerically stable coding schemes building on recent works, e.g. [35]–[37], focusing on ϵ-error instead of exact computation.

REFERENCES

[1] Full version with proofs and detailed numerical results. [Online]. Available: https://arxiv.org/abs/2105.01973

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[32] 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.

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

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

[35] 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

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

[37] 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.