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

Haewon Jeong, Ateet Devulapalli, Viveck Cadambe, Flavio Calmon

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. Moreover, we develop an optimization framework based on alternating minimization that enables the discovery of new codes for approximate matrix multiplication.

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]. 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 [26]).

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

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

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 [27]. While the results of [3], [28] 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 [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], [28] 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 also show that for PolyDot/Entangled polynomial codes [3], [28], [29] 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[2] from $p^2q + q - 1$ to p^2q by allowing ϵ-error. 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 illustrate through numerical examples that our optimization approach finds approximate computing codes with favourable trade-offs between approximation error and recovery threshold. 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], [27], 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], [28]. 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

[2]Strictly speaking, the recovery threshold of entangled polynomial codes depends on the bilinear complexity, which can be smaller than $p^2q + q - 1$ [28].

tight lower bound on the recovery threshold. These works focused on exact recovery of the computation output.

References [30]–[32] 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], [33], 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 [33] 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], [33].

A related line of work in [34], [35] study coded polynomial evaluation beyond exact recovery and note techniques to improve the quality of the approximation. References [36], [37] develops machine learning techniques for approximate learning; while they show empirical existence codes with low recovery thresholds (such as single parity codes [36]) for learning tasks they do not provide theoretical guarantees. Specifically, while [36], [37] 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 [34]–[37], 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 [38]–[41] 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 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

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}. \tag{1}$$

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

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

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[3]:

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

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

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

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:

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

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

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

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.

[3]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.

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 $K(m, \epsilon, \mathbf{f}, \mathbf{g}, \mathbf{d})$. 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 all possible decoding functions $\mathbf{d}$, i.e.,

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

Note that parameters p and q are embedded in $\mathbf{f}$ and $\mathbf{g}$ and hence $K^*(m, \epsilon)$ is the minimum over all combinations of p, q such that $pq = m$. Through an achievability scheme in [3] and a converse in [28], for exact recovery, the optimal threshold has been characterized to be $2m - 1$:

Theorem 1 (Adaptation of Theorem 2 in [28] and Theorem III.1 in [3])**.** *Under the system model given in Section II-B*

$$K^*(m, \epsilon = 0) = 2m - 1. \tag{10}$$

D. Summary of Main Result

Our main result is summarized the following theorem:

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

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

(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 $0 < \epsilon < \eta^2$, $K^*(m, \epsilon) \geq m$.

The achievability scheme given in Theorem 3 is only for $p = 1$. For a fixed $p > 1$, we propose ϵ-approximate PolyDot strategy which reduces the recovery threshold of Entangled-Poly codes from $p^2q + q - 1$ to p^2q by allowing ϵ-error in the recovered output (Theorem 5).

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 prove the converse result which states 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 show that a simple adaptation of MatDot codes can be used for approximate coded computing.

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

$$p_{\mathbf{A}}(x) = \sum_{i=1}^{m} \mathbf{A}_i x^{i-1}, p_{\mathbf{B}}(x) = \sum_{j=1}^{m} \mathbf{B}_j x^{m-j}. \tag{12}$$

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

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

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:

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

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 carefully choose the evaluation points that reduce this overhead. In fact, we select 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 $||\mathbf{A}||_F, ||\mathbf{B}||_F \le \eta$. Let $\epsilon \in \mathbb{R}$ be a constant such that*

$$0 < \epsilon < \min(2, 3\eta^2\sqrt{2m-1}). \tag{14}$$

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

$$|\lambda_i| < \frac{\epsilon}{6\eta^2\sqrt{2m-1}(m-1)m}, \quad i \in [P]. \tag{15}$$

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:*

$$K(m, \epsilon, \mathbf{f}_{\epsilon\text{-}MatDot}, \mathbf{g}_{\epsilon\text{-}MatDot}, \mathbf{d}_{\epsilon\text{-}MatDot}) = m, \tag{16}$$

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. *When* $\epsilon \geq \min(2, 3\eta^2\sqrt{2m-1})$*, we can use* ϵ'*-Approximate MatDot codes for some* $\epsilon' < \min(2, 3\eta^2\sqrt{2m-1})$*. Then,* (16) *can be expressed as:*

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

While we defer the full proof to Appendix B, 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:

$$S(x) = P(x)Q(x) + R(x), \tag{18}$$

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

$$S(\lambda_i) = R(\lambda_i). \tag{19}$$

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:

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

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:

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

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.

C. Converse

We have shown that for *any* matrices $\mathbf{A}$ and $\mathbf{B}$, and with a recovery threshold of m, ϵ-approximate 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 $0 < \epsilon < \eta^2$,*

$$K^*(m, \epsilon) \geq m. \tag{22}$$

Proof is given in Appendix C.

D. Approximate PolyDot Codes

The construction of ϵ-approximate MatDot codes achieves the optimal ϵ- approximate recovery threshold, but is limited to $p = 1, q = m$ in (3). For arbitrary p and q, the recovery threshold of $p^2q + q - 1$ is achieved by PolyDot codes (Entangled-Poly codes). In this section, we show that—similarly to MatDot Codes—the recovery threshold of Polydot codes can be improved by allowing an ϵ-approximation of the matrix multiplication and selecting evaluation points near zero. We briefly review next the construction of PolyDot codes [3] (also known as Entangled-Poly codes [28]).

Construction 3 (PolyDot (Entangled-Poly) Codes [3], [28])**.** *In [3], a general framework for PolyDot codes is proposed as follows:*

$$p_{\mathbf{A}}(x, y) = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{A}_{i,j} x^{i-1} y^{j-1}, \;\; p_{\mathbf{B}}(y, z) = \sum_{k=1}^{q} \sum_{l=1}^{p} \mathbf{B}_{k,l} y^{q-k} z^{l-1}, \tag{23}$$

where the input matrices are split as (3)*. Substituting $x = y^q$ and $z = y^{pq}$ results in Entangled-Poly codes [28]:*

$$p_{\mathbf{A}}(y) = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{A}_{i,j} y^{q(i-1)+(j-1)}, \;\; p_{\mathbf{B}}(y) = \sum_{k=1}^{q} \sum_{l=1}^{p} \mathbf{B}_{k,l} y^{q-k+pq(l-1)}. \tag{24}$$

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:

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

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

We describe next a construction for ϵ-approximate PolyDot codes.

Construction 4 (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 3 with evaluation points $\lambda_1, \ldots, \lambda_P$ that satisfy:*

$$|\lambda_i| < \min\left(\frac{\epsilon}{\eta^2 q(p^2q-1)}, \frac{1}{p^2q-1}\right), \;\; i \in [P]. \tag{25}$$

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

Theorem 5. *For any $\epsilon > 0$, the ϵ-approximate PolyDot codes in Construction 4 achieves:*

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

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

Notice that PolyDot codes are a generalized version of MatDot codes, i.e., by setting $p = 1, q = m$, Construction 3 reduces to MatDot codes. Hence, the result in Theorem 5 also applies to ϵ-approximate MatDot codes. In fact, the proof of this theorem yields a slightly improved error bound of ϵ-approximate MatDot codes given in Section III-A.

Remark 2. *The condition on the evaluation points λ_i's in Construction 2 can be relaxed to:*

$$|\lambda_i| < \min\left(\frac{\epsilon}{\eta^2 \cdot m(m-1)}, \frac{1}{m}\right), \quad i \in [P]. \tag{26}$$

The techniques used in the proofs of Theorem 3 and Theorem 4 are distinct, yet both proofs are sufficient to demonstrate the recovery threshold of ϵ-approximate MatDot codes. We present both in this paper since they may serve as blueprints for future ϵ-approximate code constructions.

IV. An Optimization Approach to Approximate Coded Computing

The Approximate MatDot code construction shows the theoretical possibility that the recovery threshold can be brought down from $2m-1$ to m. In this section, we propose another approach to find an approximate coded computing strategy. As we are not aiming for zero error, we pose the question as an optimization problem where the difference between the original matrix and the reconstructed matrix is minimized. The goal of optimization is to find ϵ such that $K^*(m, \epsilon) \leq k$ for a given k, within the space of linear encoding and decoding functions.

In Section IV-A, we illustrate our optimization framework through a simple setting of $P = 3$ nodes. In Section IV-B, we describe our formulation formally, for arbitrary values of parameters P, K, m.. We report numerical results for our optimization algorithm in Section IV-C.

A. A simple example

Consider an example of $m = 2, k = 2, P = 3$. The input matrices are split into:

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix}\mathbf{B}_1 \\ \mathbf{B}_2\end{bmatrix}. \tag{27}$$

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

$$\boldsymbol{\alpha}^{(i)} = \begin{bmatrix}\alpha_1^{(i)} \\ \alpha_2^{(i)}\end{bmatrix}, \boldsymbol{\beta}^{(i)} = \begin{bmatrix}\beta_1^{(i)} \\ \beta_2^{(i)}\end{bmatrix} \tag{28}$$

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

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

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:

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

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 the previous equality to hold for any $\mathbf{A}$ and $\mathbf{B}$, the coefficients must satisfy:

$$\begin{aligned}\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix} &= d_1 \begin{bmatrix}\alpha_1^{(1)}\beta_1^{(1)} & \alpha_1^{(1)}\beta_2^{(1)} & \alpha_2^{(1)}\beta_1^{(1)} & \alpha_2^{(1)}\beta_2^{(1)}\end{bmatrix} \\ &+ d_2 \begin{bmatrix}\alpha_1^{(2)}\beta_1^{(2)} & \alpha_1^{(2)}\beta_2^{(2)} & \alpha_2^{(2)}\beta_1^{(2)} & \alpha_2^{(2)}\beta_2^{(2)}\end{bmatrix}\end{aligned} \tag{29}$$

By reshaping the length-4 vectors in (29) into 2×2 matrices and denoting the identity matrix by $\mathbf{I}_{2\times2}$, (29) is equivalent to

$$\mathbf{I}_{2\times2} = \sum_{i=1}^{2} d_i \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}. \tag{30}$$

Encoding coefficients $\boldsymbol{\alpha}^{(i)}$'s, $\boldsymbol{\beta}^{(i)}$'s and the decoding coefficients d_i's that satisfy the equality in (30) 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 (30) 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 (30). One possible objective function for this is:

$$||\mathbf{I}_{2\times2} - \sum_{i=1}^{2} d_i \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}||_F^2. \tag{31}$$

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:

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

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:

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	$\left[\boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\beta}^{(i)}\right]_{i=1,\ldots,P}$
$\mathbf{Z}^{(p)}$	$k \times k$	$\mathbf{Z}^{(\text{full})}\big\|_{i \in \mathcal{S}_p, j \in \mathcal{S}_p}$
$\mathbf{z}^{(p)}$	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

Optimization for Approximate Coded Computing:

$$\min_{\substack{\boldsymbol{\alpha}^{(i)}, \boldsymbol{\beta}^{(i)}, \mathbf{d}^{(p)} \\ i=1,\ldots,n, \\ p=1,\ldots,\binom{P}{k}}} \sum_{p=1}^{\binom{P}{k}} ||\mathbf{I}_{m \times m} - \sum_{i \in \mathcal{S}_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)} \boldsymbol{\beta}^{(i)T}||_F^2. \tag{32}$$

Notice that (32) 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 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* (32) *satisfy*

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

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

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

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

Proof is given in Appendix A. Algorithm 1 presents an alternating minimization procedure for computing a local minimum of (32). The algorithm sequentially solves conditions (i)–(iii) in Proposition 1. Since each step corresponds to minimizing (32) 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.

We next show how the optimization objective in (32) is related to the relative error of the

Algorithm 1: Alternating Quadratic Minimization

Input: Positive Integers m, k and P ($P > k$);
Output: $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, $\mathbf{d}^{(p)}$ ($p = 1, \dots, P$);
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 $\mathbf{Y}_{\mathcal{B}} \cdot \boldsymbol{\mathcal{A}} = \operatorname{diag}(\mathbf{y}) \cdot \boldsymbol{\mathcal{B}}$;
  Compute $\mathbf{Y}_{\boldsymbol{\mathcal{A}}}$;
  $\boldsymbol{\mathcal{B}} \leftarrow \boldsymbol{\mathcal{B}}^*$, $\boldsymbol{\mathcal{B}}^*$: solution of $\mathbf{Y}_{\boldsymbol{\mathcal{A}}} \cdot \boldsymbol{\mathcal{B}} = \operatorname{diag}(\mathbf{y}) \cdot \boldsymbol{\mathcal{A}}$;
end

computation output. Let $\ell^{(p)}$ be the loss function for the p-th scenario, i.e.,

$$\ell^{(p)} = ||\mathbf{I}_{m\times m} - \sum_{i\in\mathcal{S}_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)} \boldsymbol{\beta}^{(i)T}||_F^2. \tag{33}$$

Theorem 6. *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:*

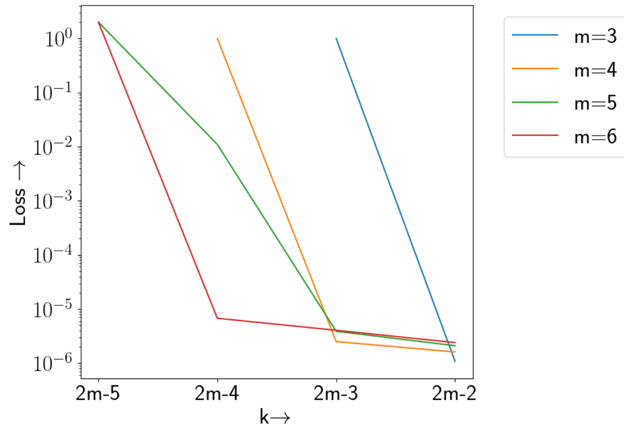
$$||\mathbf{C} - \widehat{\mathbf{C}}_{\mathcal{S}_p}||_F \leq \sqrt{\ell^{(p)}} \cdot m \cdot \eta^2. \tag{34}$$

The proof is given in Appendix E

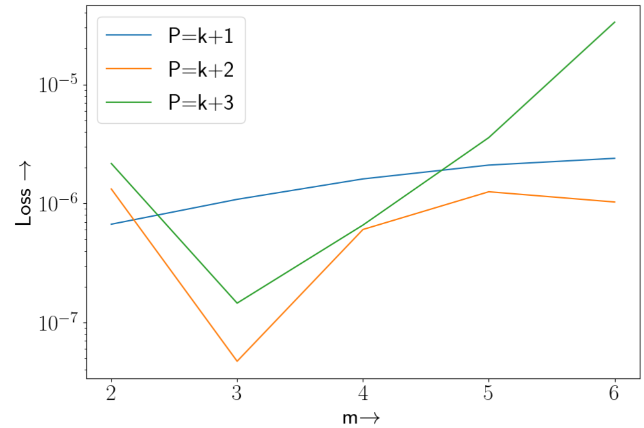
C. Optimization Results

We summarize the results of running Algorithm 1 for various combinations of parameters, m, k, P in Fig. 1. We report the best result out of 20 random initializations; for each trial, we ran Algorithm 1 for 100,000 iterations. From Fig. 1a, we see that the loss function is very small ($\sim 10^{-4}$) for $k < 2m-1$. However, the loss increases as k gets closer to m, reaching ~ 1. This could be due to the optimization algorithm converging to a local minimum as k approaches m. In Fig. 1b, we observe that the loss for $k = 2m-2$ remains small for different m values. In Fig. 1c the loss function increases as m and P get bigger because the number of terms we add in the loss function increases with m and P.

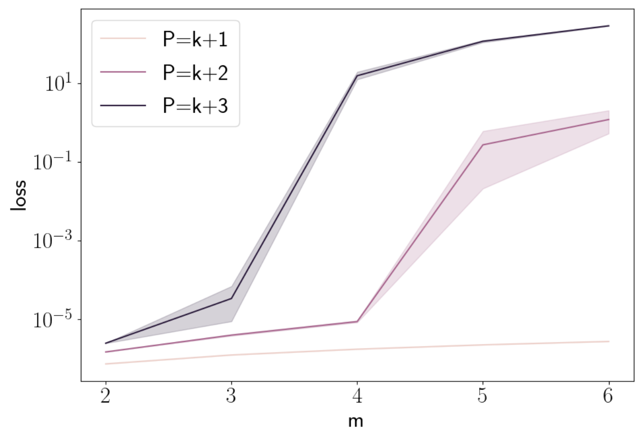
In Fig. 2, we compare the performance of the conventional MatDot and Chebyshev polynomial-based codes [38] with the approximate MatDot codes and optimization codes developed in this paper. The N_{succ} parameter represents the number of non-straggling nodes. In Fig. 2a the loss is computed in accordance with (32), where $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \mathbf{Y}$ are derived from the encoding and decoding procedures of the respective codes. In Fig. 2b, we show the actual error in the decoded matrix product, i.e., $\epsilon = ||\mathbf{C} - \mathbf{C}||_{\max}$. To compute this, we performed multiplications of two random unit matrices. Approximate MatDot codes are constructed using the evaluation points drawn iid from Unif$[-7 \times 10^{-6}, 7 \times 10^{6}]$, while MatDot codes are constructed with evaluation points drawn iid from Unif$[-0.5, 0.5]$. We observe that when $N_{\text{succ}} \geq 2m-1$, MatDot or Chebyshev



(a) Loss vs k for P=k+1



(b) Min Loss over 1000 seeds vs m for k=2m-2



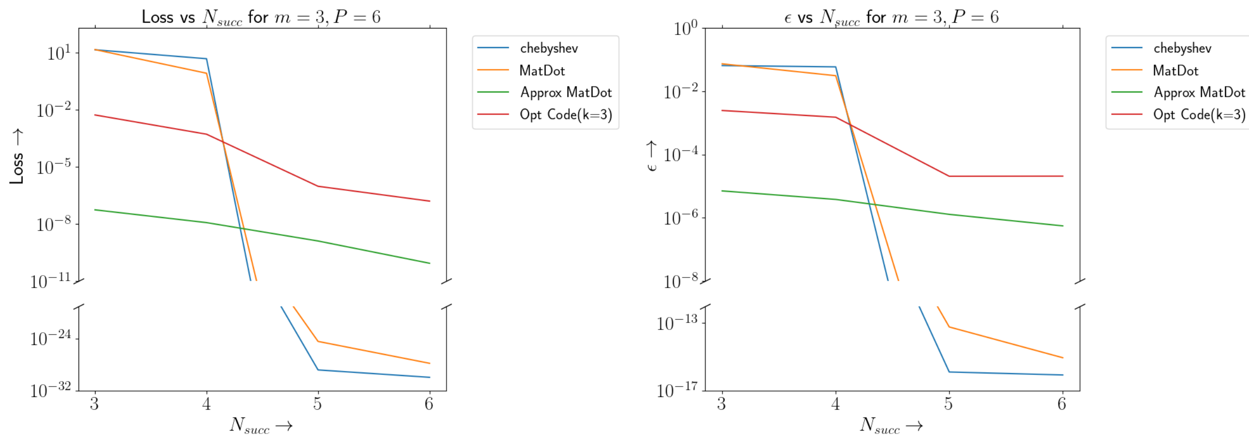
(c) Avg Loss over 1000 seeds vs m for k=2m-2 with 95% confidence interval

Figure 1: Summary of results of running Algorithm 1 for 100,000 iterations. The y-axis is the loss function given in (32).

codes perform best, and when $N_{\text{succ}} < 2m-1$, Approximate MatDot codes and optimized codes significantly outperform Chebyshev and MatDot codes. These codes are constructed for small m, i.e., $m = 3$. As we increase m, we observe that the performance of approximate codes degrades due to the numerical instability of encoding matrices.

V. Application

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.).



(a) Loss vs N_{succ}

(b) ϵ vs N_{succ}

Figure 2: Performance comparison between different coding methods, over various number of successful nodes for fixed $m = 3$ and $P = 6$. The y-axis in (a) represents the loss function given in (32) and the y-axis in (b) represents the empirical evaluation of ϵ, i.e., $||\widehat{\mathbf{C}} - \mathbf{C}||_{\max}$.

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 [42] 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 $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1) \ldots (\boldsymbol{x}_D, \boldsymbol{y}_D)\}$ 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 $\boldsymbol{Y} \in \mathbb{F}_2^{J \times D} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_D]$. Let $\boldsymbol{W} = [\boldsymbol{w}_1; \ldots; \boldsymbol{w}_J]$ ($\boldsymbol{w}_j$ is a row vector) be a matrix that comprises the logistic regression training parameters. The cross entropy loss is given by:

$$L(\boldsymbol{W}; \mathcal{D}) = \sum_{i=1}^{D} \sum_{j=1}^{J} y_{ji} \log p(y_{ji} = 1 | \boldsymbol{x}_i) \tag{35}$$

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

$$\boldsymbol{Z} = \boldsymbol{W} \boldsymbol{X} \tag{36}$$

The gradient is computed as:

$$\frac{\partial}{\partial \boldsymbol{W}} L(\boldsymbol{W}; \mathcal{D}) = \boldsymbol{H}\boldsymbol{X}^T \tag{37}$$

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 (36) and (37). In (36), 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 (37).

B. 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 [42]. A learning rate of 0.001 and batch size of 128 were used. Each experiment was run for 1,000 iterations. Table II and III show the accuracy results obtained for training and test datasets. Define loss for a particular straggler pattern by

$$\mathcal{L}(p) = ||\mathbf{I}_{m\times m} - \sum_{i\in S_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)} \boldsymbol{\beta}^{(i)T}||_F^2$$

All entries in Table II represents the accuracies for the worst case failure pattern $\text{argmax}_p\, \mathcal{L}(p)$. To simulate a realistic scenario where nodes may straggle randomly for every multiplication, we ran the experiments for random failure scenario, for which accuracies are given in Table III. For each coding scheme, we fit the corresponding encoding and decoding matrices into $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{d}^{(p)}$. For example, for approximate matdot, Vandermonde matrices are used in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and its corresponding decoding coefficients are put in $\mathbf{d}^{(p)}$. "Opt Code" represents the codes obtained from the optimization algorithm 1. The training accuracy for uncoded strategy (without failed nodes) is 80.86±0.32 and the test accuracy is 81.73±0.43.

	Training Accuracy (%)			Test Accuracy (%)		
(m, N_{succ}, P)	Approx MatDot	Chebyshev	Opt Code	Approx MatDot	Chebyshev	Opt Code
(5,5,2)	80.12±0.65	53.66±6.69	52.32±11.37	81.01±0.71	54.18±6.79	52.77±11.59
(5,6,2)	80.85±0.33	57.81±6.19	54.82±15.56	81.73±0.43	58.30±6.28	55.32±15.79
(5,7,2)	80.86±0.32	62.81±5.48	66.81±11.29	81.73±0.43	63.18±5.62	67.48±11.50
(5,8,2)	80.85±0.32	58.50±6.83	78.12±7.70	81.73±0.43	59.03±6.88	78.95±7.86
(5,9,2)	80.85±0.32	80.85±0.32	80.85±0.33	81.73±0.43	81.73±0.43	81.73±0.43
(20,20, 2)	9.69±2.92	55.99±9.05	72.96±5.96	9.70±3.01	56.72±9.23	73.73±6.08
(50,50, 2)	9.69±2.91	58.70±6.68	—	9.70±3.00	59.14±6.80	—

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

VI. 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

(m, N_{succ}, P)	Training Accuracy (%) Approx MatDot	Training Accuracy (%) Chebyshev	Training Accuracy (%) Opt Code	Test Accuracy (%) Approx MatDot	Test Accuracy (%) Chebyshev	Test Accuracy (%) Opt Code
(5,5,2)	80.73±0.42	82.48±0.45	79.47±9.38	81.60±0.50	83.35±0.53	80.36±9.51
(5,6,2)	80.85±0.33	82.47±0.42	74.55±17.14	81.73±0.43	83.33±0.49	75.35±17.35
(5,7,2)	80.86±0.32	82.15±0.41	79.94±5.00	81.73±0.43	83.00±0.49	80.81±5.08
(5,8,2)	80.85±0.32	81.70±0.38	80.86±0.53	81.73±0.43	82.56±0.45	81.73±0.59
(5,9,2)	80.85±0.32	80.85±0.32	80.85±0.33	81.73±0.43	81.73±0.43	81.73±0.43
(20,20, 2)	9.81±2.97	82.21±0.36	81.94±4.64	9.82±3.06	83.09±0.44	82.83±4.69
(50,50, 2)	9.69±2.91	60.34±6.72	—	9.70±3.00	60.76±6.85	—

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

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. [38]–[40], 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.

[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] 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

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

[28] 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.

[29] 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.

[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] 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.

[33] 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.

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

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

[36] J. Kosaian, K. 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.

[37] ——, “Learning-based coded computation,” *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

[38] 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

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

[40] 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.

[41] 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.

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

[43] 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 PROPOSITION 1

For simple demonstration, let us focus on the case where $\mathcal{S}_p = [1, \dots, k]$ and expand the term inside the sum. In the following equations, we will omit the superscript (p) for simplification.

$$
\begin{aligned}
||\mathbf{I}_{m\times m} - \sum_{i\in\mathcal{S}_p} d_i \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}||_F^2 &= \mathrm{Tr}\left(\left(\mathbf{I} - \sum_{i=1}^{k} d_i\mathbf{X}_i\right)^T \left(\mathbf{I} - \sum_{i=1}^{k} d_i\mathbf{X}_i\right)\right) \quad (\mathbf{X}_i = \boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}) \\
&= \mathrm{Tr}\left(\mathbf{I} - \sum_{i=1}^{k} d_i\mathbf{X}_i^T - \sum_{i=1}^{k} d_i\mathbf{X}_i + \left(\sum_{i=1}^{k} d_i\mathbf{X}_i\right)^T \left(\sum_{j=1}^{k} d_j\mathbf{X}_j\right)\right) \\
&= m - 2\sum_{i=1}^{k} d_i \,\mathrm{Tr}(\mathbf{X}_i) + \sum_{i=1}^{k}\sum_{j=1}^{k} d_i d_j \,\mathrm{Tr}(\mathbf{X}_i^T\mathbf{X}_j) \\
&= m - 2\sum_{i=1}^{k} d_i \boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\beta}^{(i)} + \sum_{i=1}^{k}\sum_{j=1}^{k} d_i d_j (\boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\alpha}^{(j)})(\boldsymbol{\beta}^{(i)} \cdot \boldsymbol{\beta}^{(j)}) \\
&= m - 2\mathbf{d}\cdot\mathbf{z} + \mathbf{d}^T\mathbf{Z}\mathbf{d},
\end{aligned}
$$

where $\mathbf{d}$ and $\mathbf{z}$ are length-k column vectors: $\mathbf{d} = [d_1, \dots, d_k]$ and $\mathbf{z} = [\boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\beta}^{(i)}]_{i=1,\dots,k}$. $\mathbf{Z}$ is a $k \times k$ matrix: $\mathbf{Z} = (\boldsymbol{\mathcal{A}}_k^T\boldsymbol{\mathcal{A}}_k) \odot (\boldsymbol{\mathcal{B}}_k^T\boldsymbol{\mathcal{B}}_k)$, where $\boldsymbol{\mathcal{A}}_k = \begin{bmatrix}\boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \dots & \boldsymbol{\alpha}_k\end{bmatrix}$ and $\boldsymbol{\mathcal{B}}_k = \begin{bmatrix}\boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \dots & \boldsymbol{\beta}_k\end{bmatrix}$.

The partial derivative with respect to $\mathbf{d}$ can be represented as: $\frac{\partial}{\partial\mathbf{d}}L = -2\mathbf{z} + 2\mathbf{Z}\mathbf{d}$. Thus, the optimal $\mathbf{d}^*$ can be obtained by solving: $\mathbf{Z}\mathbf{d} = \mathbf{z}$. Note that this can be easily generalized to any $\mathbf{d}^{(p)} = \begin{bmatrix}d_i\end{bmatrix}_{i\in\mathcal{S}_p}$. It only requires using different $\mathbf{Z}$ and $\mathbf{z}$ as follows:

$$\mathbf{Z} = (\boldsymbol{\mathcal{A}}^{(p)T}\boldsymbol{\mathcal{A}}^{(p)}) \odot (\boldsymbol{\mathcal{B}}^{(p)T}\boldsymbol{\mathcal{B}}^{(p)}), \;\; \mathbf{z}^{(p)} = [\boldsymbol{\alpha}^{(i)} \cdot \boldsymbol{\beta}^{(i)}]_{i\in S_p},$$

where $\boldsymbol{\mathcal{A}}^{(p)} = [\boldsymbol{\alpha}_i]_{i\in S_p}$, $\boldsymbol{\mathcal{B}}^{(p)} = [\boldsymbol{\beta}_i]_{i\in S_p}$.

To obtain the gradient with respect to $\boldsymbol{\alpha}^{(i)}$'s and $\boldsymbol{\beta}^{(i)}$'s, let us expand the loss function given in (32) again. We now want to include the outer sum:

$$\begin{aligned}
L &= \sum_{p=1,\dots,\binom{n}{k}} ||\mathbf{I}_{m\times m} - \sum_{i\in\mathcal{S}_p} d_i^{(p)}\boldsymbol{\alpha}^{(i)}\boldsymbol{\beta}^{(i)T}||_F^2 \\
&= \sum_{p=1,\dots,\binom{n}{k}} \Big(m - 2\sum_{i\in\mathcal{S}_p} d_i^{(p)}\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\beta}^{(i)} + \sum_{i\in\mathcal{S}_p}\sum_{j\in\mathcal{S}_p} d_i^{(p)}d_j^{(p)}(\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\alpha}^{(j)})(\boldsymbol{\beta}^{(i)}\cdot\boldsymbol{\beta}^{(j)})\Big) \\
&= \binom{n}{k}\cdot m - 2\sum_{i\in[n]}\big(\sum_{p:i\in\mathcal{S}_p} d_i^{(p)}\big)\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\beta}^{(i)} + \sum_{i\in[n]}\sum_{j\in[n]}\big(\sum_{p:i,j\in\mathcal{S}_p} d_i^{(p)}d_j^{(p)}\big)(\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\alpha}^{(j)})(\boldsymbol{\beta}^{(i)}\cdot\boldsymbol{\beta}^{(j)}) \\
&= \binom{n}{k}\cdot m - 2\sum_{i\in[n]} y_i\cdot\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\beta}^{(i)} + \sum_{i\in[n]}\sum_{j\in[n]} Y_{i,j}(\boldsymbol{\alpha}^{(i)}\cdot\boldsymbol{\alpha}^{(j)})(\boldsymbol{\beta}^{(i)}\cdot\boldsymbol{\beta}^{(j)})
\end{aligned}$$

In the last line, we let $y_i = \sum_{p:i\in\mathcal{S}_p} d_i^{(p)}$ and $Y_{i,j} = \sum_{p:i,j\in\mathcal{S}_p} d_i^{(p)}d_j^{(p)}$. Now, the gradient of L with respect to $\boldsymbol{\alpha}^{(i)}$ can be written as:

$$\frac{\partial}{\partial\boldsymbol{\alpha}^{(i)}}L = -2y_i\boldsymbol{\beta}^{(i)} + 2\sum_{j\in[n]} Y_{i,j}(\boldsymbol{\beta}^{(i)}\cdot\boldsymbol{\beta}^{(j)})\boldsymbol{\alpha}^{(j)}. \tag{38}$$

Following the notation that $\boldsymbol{\mathcal{A}} = \begin{bmatrix}\boldsymbol{\alpha}^{(1)} & \dots & \boldsymbol{\alpha}^{(n)}\end{bmatrix}$ and $\boldsymbol{\mathcal{B}} = \begin{bmatrix}\boldsymbol{\beta}^{(1)} & \dots & \boldsymbol{\beta}^{(n)}\end{bmatrix}$, this can be written in a matrix form: $\frac{\partial}{\partial\boldsymbol{\mathcal{A}}}L = -2\mathrm{diag}(\mathbf{y})\boldsymbol{\mathcal{B}}^T + 2\mathbf{Y}_{\boldsymbol{\mathcal{B}}}\boldsymbol{\mathcal{A}}^T$, where $\mathbf{y} = [y_i]_{i=1,\dots,n}$ is a column vector of length n and $\mathbf{Y}_{\mathcal{B}} = [Y_{i,j}(\boldsymbol{\beta}^{(i)}\cdot\boldsymbol{\beta}^{(j)})]_{i,j=1,\dots,n} = \mathbf{Y}\odot(\boldsymbol{\mathcal{B}}^T\boldsymbol{\mathcal{B}})$ is an $n\times n$ matrix. Thus, the optimal $\boldsymbol{\mathcal{A}}^*$ can be obtained by solving: $\mathbf{Y}_{\mathcal{B}}\cdot\boldsymbol{\mathcal{A}} = \mathrm{diag}(\mathbf{y})\cdot\boldsymbol{\mathcal{B}}$. Similarly, the optimal $\boldsymbol{\mathcal{B}}^*$ can be obtained by solving: $\mathbf{Y}_{\mathcal{A}}\cdot\boldsymbol{\mathcal{B}} = \mathrm{diag}(\mathbf{y})\cdot\boldsymbol{\mathcal{A}}$, where $\mathbf{Y}_{\mathcal{A}} = \mathbf{Y}\odot(\boldsymbol{\mathcal{A}}^T\boldsymbol{\mathcal{A}})$.

Appendix B
Proof of Theorem 3

Let f be a $(k-1)$-degree polynomial $f(x) = a_0 + a_1x + \cdots + a_{k-1}x^{k-1}$, and we use $\mathsf{vec}(f)$ to denote the row vector representation of the coefficients of f, i.e.,

$$\mathsf{vec}(f) \triangleq \begin{bmatrix}a_0 & a_1 & \cdots & a_{k-1}\end{bmatrix} \triangleq \mathbf{a}. \tag{39}$$

Also, let $\boldsymbol{\lambda} \triangleq [\lambda_j]_{j\in[m]} \in \mathbb{R}^m$ be m distinct evaluation points. We define a Vandermonde matrix for $\boldsymbol{\lambda}$ of degree $k-1$ as:

$$\mathsf{Vander}(\boldsymbol{\lambda}, k) = \begin{bmatrix} 1 & \cdots & 1 \\ \lambda_1 & \cdots & \lambda_m \\ \vdots & \ddots & \vdots \\ \lambda_1^{k-1} & \cdots & \lambda_m^{k-1}\end{bmatrix}_{k\times m}. \tag{40}$$

The evaluations of f at the points $\boldsymbol{\lambda}$ can be written as

$$\begin{bmatrix}f(\lambda_1) & f(\lambda_2) & \cdots & f(\lambda_m)\end{bmatrix} = \mathbf{a}\cdot\mathbf{V}, \tag{41}$$

where $\mathbf{V} = \mathsf{Vander}(\boldsymbol{\lambda}, k)$. When $m < k$, the null space of $\mathbf{V}$ can be conveniently expressed in terms of elementary symmetric polynomials.

Definition 1 (Elementary Symmetric Polynomial)**.** Let $\mathbf{x} = (x_1, \cdots, x_n)$. For $l \in \{0, 1, \ldots, n\}$, the elementary symmetric polynomials in n variables $e_l : \mathbb{R}^n \to \mathbb{R}$ are given by

$$e_l(\mathbf{x}) \triangleq \begin{cases} \sum\limits_{\substack{S \subseteq [n] \\ |S| = l}} \prod\limits_{i \in S} x_i, & \text{if } l = 1, \ldots, n, \\ 1, & \text{if } l = 0. \end{cases} \tag{42}$$

In particular, $e_1(\mathbf{x}) = \sum_{i \in [n]} x_i$ and $e_n(\mathbf{x}) = \prod_{i \in [n]} x_i$.

Lemma 1. *For $m < k$, the left null space of $\mathbf{V}$ is spanned by $\{\mathbf{u}_i\}_{i \in [k-m]} \subset \mathbb{R}^k$, where $\mathbf{u}_i = \mathsf{vec}(p_i)$ for the polynomials p_i's defined as:*

$$p_i(x) \triangleq x^{i-1} \prod_{j=1}^{m} (x - \lambda_j). \tag{43}$$

Proof. First, note that $\mathbf{u}_i \in \mathsf{null}(\mathbf{V}^T)$: $\mathbf{u}_i \cdot \mathbf{V} = \begin{bmatrix} p_i(\lambda_1) & \cdots & p_i(\lambda_m) \end{bmatrix} = \mathbf{0}$. Next, we show that $\mathsf{dim}(\mathsf{span}(\mathbf{u}_1, \cdots, \mathbf{u}_{k-m})) = k - m$. The coefficients of the Lagrange polynomial $p_1(x) = \prod_{j=1}^{m} (x - \lambda_j)$ can be written as:

$$\mathbf{u}_1 = \begin{bmatrix} e_m(\boldsymbol{\lambda}) & \cdots & e_0(\boldsymbol{\lambda}) & 0 & 0 & \cdots 0 \end{bmatrix}, \tag{44}$$

with $k - m - 1$ trailing zeros, and

$$\begin{aligned} \mathbf{u}_2 &= \begin{bmatrix} 0 & e_m(\boldsymbol{\lambda}) & \cdots & e_0(\boldsymbol{\lambda}) & 0 & \cdots 0 \end{bmatrix}, \\ &\vdots \\ \mathbf{u}_{k-m} &= \begin{bmatrix} 0 & 0 & \cdots & 0 & e_m(\boldsymbol{\lambda}) & \cdots & e_0(\boldsymbol{\lambda}) \end{bmatrix}. \end{aligned} \tag{45}$$

Let $\mathbf{U} \in \mathbb{R}^{(k-m) \times k}$ be the matrix obtained by concatenating $\mathbf{u}_i$ row-wise, i.e., the matrix with i-th row equal to $\mathbf{u}_i$. Note that $\mathbf{U} = [\mathbf{U}_1 \ \mathbf{U}_2]$, where $\mathbf{U}_2 \in \mathbb{R}^{(k-m) \times (k-m)}$ is a lower-triangular matrix with diagonal entries equal to $e_0(\boldsymbol{\lambda}) = 1$. Consequently, $\mathbf{U}_2$ is full-rank (in particular, $\det(\mathbf{U}_2) = 1$). Therefore $\mathbf{U}$ is also full-rank and $\mathsf{rank}(\mathbf{U}) = \mathsf{dim}(\mathsf{span}(\mathbf{u}_1, \cdots, \mathbf{u}_{k-m})) = k - m$. $\square$

We prove next a bound on the evaluation of the elementary symmetric polynomials $e_l, l \neq 0$ in terms of the ℓ_∞-norm of its entries.

Lemma 2. *Let $0 < \epsilon \leq 2$ and $\mathbf{x} \in \mathbb{R}^n$. If $\|\mathbf{x}\|_\infty \leq \epsilon / n$, then $|e_l(\mathbf{x})| \leq \epsilon$ for $l \in \{1, 2, \ldots, n\}$.*

Proof.

$$\begin{aligned} |e_l(\mathbf{x})| &= \Big| \sum_{\substack{S \in [n] \\ |S| = l}} \prod_{j \in S} x_j \Big| \leq \sum_{\substack{S \in [n] \\ |S| = l}} \Big| \prod_{j \in S} x_j \Big| \leq \sum_{\substack{S \in [n] \\ |S| = l}} \left(\frac{\epsilon}{n} \right)^l = \binom{n}{l} \cdot \left(\frac{\epsilon}{n} \right)^l \\ &\leq \frac{n^l}{l!} \cdot \frac{\epsilon^l}{n^l} = \frac{\epsilon^l}{l!} = \epsilon \cdot \prod_{k=2}^{l} \frac{\epsilon}{k} \leq \epsilon. \end{aligned}$$

$\square$

If $m = k$, the coefficients of f can be recovered *exactly* from $\{f(\lambda_j)\}_{j\in[m]}$ by inverting the linear system (41) as long as the evaluation points are distinct. When $m < k$ then, in general, $\mathbf{a}$ cannot be recovered exactly. In this case, the system (41) is undetermined: denoting the true (but unknown) coefficient vector by $\mathbf{a}^*$, any vector in the set $\{\mathbf{a}\}+\mathsf{null}(\mathbf{V}^T) = \{\mathbf{a}+\mathbf{n} \mid \mathbf{n} \in \mathsf{null}(\mathbf{V}^T)\}$ will be consistent with the m evaluation points. Nevertheless, we show next that if the coefficients of f have bounded norm, i.e., $\mathbf{a} \in \mathcal{B}_R \triangleq \{\mathbf{x} \in \mathbb{R}^k \text{ s.t. } ||\mathbf{x}||_2 \leq R\}$, then the first m coefficients $a_0, \ldots, a_{m-1}$ can be approximated with *arbitrary precision* by computing f at m distinct and sufficiently small evaluation points. This result is formally stated in Corollary 1, which is the main tool for proving the approximate coded computing recovery threshold.

Theorem 7. *Let $\boldsymbol{\lambda}$ be $m < k$ distinct evaluation points with corresponding $k \times m$ Vandermonde matrix $\mathbf{V} = \mathsf{Vander}(\boldsymbol{\lambda}, k-1)$ and $0 < \epsilon \leq \min(2, 3R)$. If $\|\boldsymbol{\lambda}\|_\infty < \frac{\epsilon}{3R(k-m)m}$, then for any $\mathbf{x} \in \mathcal{B}_R \cap \mathsf{null}(\mathbf{V}^T)$,*

$$|x[i]| \leq \epsilon \quad \textit{for} \quad i \in [m]. \tag{46}$$

Proof. Since $\mathbf{x} \in \mathsf{null}(\mathbf{V}^T)$, we can express $\mathbf{x}$ as:

$$\mathbf{x} = \alpha_1 \mathbf{u}_1 + \cdots + \alpha_{k-m}\mathbf{u}_{k-m}, \tag{47}$$

for some $\alpha_1, \cdots, \alpha_{k-m} \in \mathbb{R}$. For a shorthand notation, we will use e_l for $e_l(\boldsymbol{\lambda})$, and let $e_l = 0$ if $l < 0$ or $l > m$. By substituting (44), (45) into (47), we get: $x[i] = \sum_{j=1}^{k-m} \alpha_j e_{m-i+j}$ for $i = 1, \ldots, k$. Since $e_0 = 1$, $x[k] = \alpha_{k-m}e_0 = \alpha_{k-m}$. Furthermore, because $\mathbf{x} \in \mathcal{B}_R$, $|x[k]| = |\alpha_{k-m}| \leq R$. Similarly, $x[k-1] = \alpha_{k-m-1}e_0 + \alpha_{k-m}e_1 = \alpha_{k-m-1} + \alpha_{k-m}e_1$.

Now note that from Lemma 2, $|e_l(\boldsymbol{\lambda})| \leq \frac{\epsilon}{3R(k-m)} \triangleq \delta$. for $l \in [m]$. Thus,

$$|\alpha_{k-m-1}| = |x[k-1] + \alpha_{k-m}e_1| \leq |x[k-1]| + |\alpha_{k-m}e_1| \leq R + R\cdot\delta = R(1+\delta).$$

By repeating the same argument up to $x[m+1]$, for $l = 1, \ldots, k-m$, we obtain:

$$|\alpha_l| \leq R(1+\delta)^{k-m-l} \leq R(1+\frac{1}{k-m})^{k-m-l} \leq R(1+\frac{1}{k-m})^{k-m} \leq 3R. \tag{48}$$

The second inequality follows from the assumption that $\epsilon \leq 3R$ and the last inequality holds because for a positive integer n: $(1+\frac{1}{n})^n \leq 3 - \frac{1}{n} < 3$.

Now, for $i \in [m]$, $x[i]$ can be written as:

$$|x[i]| = |\sum_{j=1}^{k-m} \alpha_j e_{m-i+j}| \leq \sum_{j=1}^{k-m} |\alpha_j||e_{m-i+j}| \leq \sum_{j=1}^{k-m} |\alpha_j|\delta \leq (k-m)\cdot 3R\cdot\delta = \epsilon. \tag{49}$$

The inequality in (49) holds because $m - l + j \neq 0$ for $l \in [m]$, and thus $|e_{m-l+j}| \leq \delta$. $\square$

Corollary 1. *Consider a set $\{f(\lambda_j)\}_{j\in[m]}$ of $m < k$ evaluations of f at distinct points $\boldsymbol{\lambda}$. If $0 < \epsilon < \min(2, 3R)$ and $\|\boldsymbol{\lambda}\|_\infty < \frac{\epsilon}{6R(k-m)m}$, then for any two coefficient vectors $\mathbf{a}, \mathbf{b} \in \mathcal{B}_R$ that satisfy* (41) *(i.e., that are consistent with the evaluations), we have*

$$|a[i] - b[i]| \leq \epsilon \textit{ for } i \in [m]. \tag{50}$$

Proof. Under the assumptions of the corollary, $\mathbf{n} \triangleq \mathbf{a} - \mathbf{b} \in \mathsf{null}(\mathbf{V}^T)$. Moreover, the triangle inequality yields: $\|\mathbf{n}\| \leq \|\mathbf{a}\| + \|\mathbf{b}\| \leq 2R$. I.e., $\mathbf{n} \in \mathcal{B}_{2R} \cap \mathsf{null}(\mathbf{V}^T)$. The result follows by a direct application of Theorem 7. $\square$

For an n-by-n matrix $\mathbf{C}$, the polynomial $p_{\mathbf{C}}(x)$ is essentially a set of n^2 polynomials, having one polynomial for each $C[i,j]$ ($i,j \in [n]$). For decoding, we have to interpolate each of those n^2 polynomials. Let us denote $p_{C[i,j]}(x)$ as the (i,j)-th polynomial for $C[i,j]$ and let the row vector representation of the coefficients of $p_{C[i,j]}(x)$ as $\mathbf{p}_{[i,j]}$.

Lemma 3. *For $p_{\mathbf{C}}(x)$ given in Construction 1, the norm of $\mathbf{p}_{[i,j]} = \mathsf{vec}(p_{C[i,j]})$ is bounded as:*

$$||\mathbf{p}_{[i,j]}||_2 \leq \sqrt{2m-1}\eta^2, \tag{51}$$

if $||\mathbf{A}||_F \leq \eta$ and $||\mathbf{B}||_F \leq \eta$.

Proof. Throughout the proof, $||\cdot||$ denotes a Frobenius norm for a matrix and a 2-norm for a vector. Let $\mathbf{P}_l$ be the coefficient of x^{l-1} in $p_{\mathbf{C}}(x)$ for $l \in [2m-1]$, which can be written as:

$$\mathbf{P}_l = \sum_{\substack{1\leq i,j\leq m \\ j-i=m-l}} \mathbf{A}_i\mathbf{B}_j = \begin{cases} \sum_{1\leq i\leq l} \mathbf{A}_i\mathbf{B}_{i+m-l}, & \text{if } l \leq m \\ \sum_{l+1-m\leq i\leq m} \mathbf{A}_i\mathbf{B}_{i+m-l}, & \text{otherwise.} \end{cases}$$

Let us focus on the case when $l \leq m$ as the argument extends naturally for $l > m$. For $l \leq m$, $\mathbf{P}_l$ can be rewritten as: $\mathbf{P}_l = \sum_{1\leq i\leq l} \mathbf{A}_i\mathbf{B}_{i+m-l} = \begin{bmatrix}\mathbf{A}_1 & \cdots & \mathbf{A}_l\end{bmatrix} \cdot \begin{bmatrix}\mathbf{B}_{m-l+1} \\ \vdots \\ \mathbf{B}_m\end{bmatrix}$. As these matrices are submatrices of $\mathbf{A}$ and $\mathbf{B}$,

$$||\begin{bmatrix}\mathbf{A}_1 & \cdots & \mathbf{A}_l\end{bmatrix}|| \leq \eta, \quad \left\|\begin{bmatrix}\mathbf{B}_{m-l+1} \\ \vdots \\ \mathbf{B}_m\end{bmatrix}\right\| \leq \eta. \tag{52}$$

Since $||\mathbf{X}\mathbf{Y}|| \leq ||\mathbf{X}|| \cdot ||\mathbf{Y}||$, we have $||\sum_{1\leq i\leq l} \mathbf{A}_i\mathbf{B}_{i+m-l}|| \leq \eta^2$. We can apply the same argument for $l > m$ and show: $||\mathbf{P}_l|| = ||\sum_{\substack{1\leq i,j\leq m \\ j-i=m-l}} \mathbf{A}_i\mathbf{B}_j|| \leq \eta^2 \quad$ for $l \in [2m-1]$. Finally,

$$||\mathbf{p}_{[i,j]}|| = \left\|\begin{bmatrix}P_1[i,j] & P_2[i,j] & \cdots & P_{2m-1}[i,j]\end{bmatrix}\right\| = \sqrt{\sum_{l=1}^{2m-1} P_l[i,j]^2} \leq \sqrt{\sum_{l=1}^{2m-1} ||\mathbf{P}_l||^2} \tag{53}$$

$$\leq \sqrt{2m-1}\eta^2. \tag{54}$$

$\square$

Algorithm 1 (Decoding of Approximate MatDot codes)**.** *Let $\boldsymbol{\lambda}^{(succ)}$ be a length-K vector with evaluation points at K successful worker nodes: $\boldsymbol{\lambda}^{(succ)} = \begin{bmatrix}\lambda_{i_1}, \cdots, \lambda_{i_K}\end{bmatrix}$, and let $\mathbf{V}^{(succ)} =$* $\mathsf{Vander}(\boldsymbol{\lambda}^{(succ)}, 2m-2)$. *Finally, we denote $\mathbf{y}_{[i,j]}^{(succ)}$ as the evaluations of $p_{C[i,j]}(x)$ at $\boldsymbol{\lambda}^{(succ)}$, i.e., $\mathbf{y}_{[i,j]}^{(succ)} = \begin{bmatrix}\widetilde{C}_{i_1}[i,j] & \cdots \widetilde{C}_{i_K}[i,j]\end{bmatrix}$. For decoding $C[i,j]$, we solve the following optimization:*

$$\widehat{\mathbf{a}} = \underset{\mathbf{a}\mathbf{V}^{(succ)}=\mathbf{y}_{[i,j]}^{(succ)}}{\operatorname{argmin}} ||\mathbf{a}||_2. \tag{55}$$

If $||\widehat{\mathbf{a}}||_2 > \sqrt{2m-1}\eta^2$*, declare failure. Otherwise,* $\widehat{C}[i,j] = \widehat{a}[m]$.

Proof of Theorem 3: First, note that the solution of the equation (55) and the true polynomial coefficients $\mathbf{p}_{[i,j]}$ both lie in $\mathcal{B}_{\sqrt{2m-1}\eta^2}$. As $||\lambda^{(\text{succ})}||_\infty < \frac{\epsilon}{6\eta^2\sqrt{2m-1}(m-1)m}$, by construction, Corollary 1 gives: $|\widehat{a}[l] - p_{[i,j]}[l]| \leq \epsilon \quad \text{for } l \in [m]$. Hence, $|\widehat{C}[i,j] - C[i,j]| = |\widehat{a}[m] - p_{[i,j]}[m]| \leq \epsilon$.

APPENDIX C

Proof of Theorem 4. We show a contradiction, i.e., assume $K(m,\epsilon) = m-1, \ \forall \epsilon < \eta^2$. We need to show that there exist matrices $\mathbf{A}, \mathbf{B}$ such that $\epsilon \geq \eta^2$ for a recovery threshold of $m-1$.

Consider f_i and g_i defined in the system model (4) and (5). Let $\boldsymbol{f} = \{f_i\}_{i=1}^P$ and $\boldsymbol{g} = \{g_i\}_{i=1}^P$ be encoding functions for $\mathbf{A}$ and $\mathbf{B}$ respectively. Consider any set $\mathcal{S}$ of $m-1$ nodes. Let $\boldsymbol{f}_{\mathcal{S}}$, $\boldsymbol{g}_{\mathcal{S}}$ denote the restriction of $\boldsymbol{f}$, $\boldsymbol{g}$ to the nodes corresponding to $\mathcal{S}$ respectively. Let $\boldsymbol{C_i} =$ output of i^{th} node, $i \in \mathcal{S}$.

$$\boldsymbol{C_i} = (f_i(\{\mathbf{A}_{j,k}\}_{j=1,k=1}^{p,q}))\ (g_i(\{\mathbf{B}_{j,k}\}_{j=1,k=1}^{q,p}))$$

Let $\boldsymbol{C} = \{\boldsymbol{C_i}\}_{i\in\mathcal{S}}$. Let $d_{\mathcal{S}}(\,\cdot\,; \boldsymbol{f}_{\mathcal{S}}, \boldsymbol{g}_{\mathcal{S}})$ denote any decoding function corresponding to the $m-1$ nodes in $\mathcal{S}$ that takes $\boldsymbol{C}$ and gives an estimate of $\mathbf{AB}$. To show a contradiction, we show that there exist matrices $\mathbf{A}, \mathbf{B}$, such that $||d_{\mathcal{S}}(\boldsymbol{C}) - \mathbf{AB}||_F \geq \eta^2 > 0$ Let $\textbf{vectorize}(\cdot)$ be a function that outputs a column-wise vectorization of the input matrix. Let $\boldsymbol{Q} \in \mathbb{R}^{p\times q}$ such that $\sigma_{\max}(\boldsymbol{Q}) = 1$ and, $\textbf{vectorize}(\boldsymbol{Q})$ is a null vector of $\boldsymbol{f}_{\mathcal{S}}$ i.e., $\boldsymbol{f}_{\mathcal{S}}(\textbf{vectorize}(\boldsymbol{Q}) \otimes \boldsymbol{D}) = \mathbf{0}$, $\forall \boldsymbol{D} \in \mathbb{R}^{\frac{n}{p}\times\frac{n}{q}}$. Note that we can scale any such $\boldsymbol{Q}$, such that its maximum singular value is 1. Let $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ be some constant matrices with bounded frobenius norms. We set

$$\mathbf{A} = \boldsymbol{Q} \otimes \bar{\mathbf{A}},\ \bar{\mathbf{A}} \in \mathbb{R}^{\frac{n}{p}\times\frac{n}{q}} \ \text{ and } \ \mathbf{B}^{(b)} = b\boldsymbol{Q}^T \otimes \bar{\mathbf{B}},\ \bar{\mathbf{B}} \in \mathbb{R}^{\frac{n}{q}\times\frac{n}{p}},\ b \in \mathbb{R}\backslash\{0\}$$

Note: $\mathbf{AB}^{(b)} = b\bar{\mathbf{A}}\bar{\mathbf{B}}$. Also observe that, $||\bar{\mathbf{A}}||_F \leq \eta$ and $||\bar{\mathbf{B}}||_F \leq \frac{\eta}{|b|}$.

Let $\boldsymbol{C}_i^{(b)} = (f_i(\{\mathbf{A}_{j,k}\}_{j=1,k=1}^{p,q}))\ (g_i(\{\mathbf{B}_{j,k}^{(b)}\}_{j=1,k=1}^{q,p}))$, $\boldsymbol{C}^{(b)} = \{\boldsymbol{C}_i^{(b)}\}_{i\in\mathcal{S}}$.

By construction $\boldsymbol{C}^{(b)} = \mathbf{0} \implies d_{\mathcal{S}}(\boldsymbol{C}^{(1)}) = d_{\mathcal{S}}(\boldsymbol{C}^{(-1)}) = d_{\mathcal{S}}(\mathbf{0})$. Then by triangle inequality,

$$||d_{\mathcal{S}}(\boldsymbol{C}^{(1)}) - \mathbf{AB}^{(1)}||_F + ||d_{\mathcal{S}}(\boldsymbol{C}^{(-1)}) - \mathbf{AB}^{(-1)}||_F \geq ||\mathbf{AB}^{(1)} - \mathbf{AB}^{(-1)}||_F \geq 2||\mathbf{AB}^{(1)}||_F$$

$$\max(||d_{\mathcal{S}}(\mathbf{0}) - \mathbf{AB}||_F, ||d_{\mathcal{S}}(\mathbf{0}) + \mathbf{AB}||_F) \geq ||\mathbf{AB}^{(1)}||_F$$

$$\begin{aligned}
||\mathbf{AB}^{(1)}||_F &= ||(\boldsymbol{Q} \otimes \bar{\mathbf{A}})(\boldsymbol{Q}^T \otimes \bar{\mathbf{B}})||_F = ||(\boldsymbol{Q}\boldsymbol{Q}^T) \otimes (\bar{\mathbf{A}}\bar{\mathbf{B}})||_F \\
&= \sqrt{\mathrm{Tr}((\boldsymbol{Q}\boldsymbol{Q}^T \otimes \bar{\mathbf{B}}^T\bar{\mathbf{A}}^T)(\boldsymbol{Q}\boldsymbol{Q}^T \otimes \bar{\mathbf{A}}\bar{\mathbf{B}}))} = \sqrt{\mathrm{Tr}((\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{Q}^T) \otimes (\bar{\mathbf{B}}^T\bar{\mathbf{A}}^T\bar{\mathbf{A}}\bar{\mathbf{B}}))} \\
&= \sqrt{\mathrm{Tr}(\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{Q}^T)}\sqrt{\mathrm{Tr}(\bar{\mathbf{B}}^T\bar{\mathbf{A}}^T\bar{\mathbf{A}}\bar{\mathbf{B}})} \\
&= ||\boldsymbol{Q}\boldsymbol{Q}^T||_F ||\bar{\mathbf{A}}\bar{\mathbf{B}}||_F
\end{aligned}$$

We can find matrices $\bar{\mathbf{A}}, \bar{\mathbf{B}}$ such that $||\bar{\mathbf{A}}\bar{\mathbf{B}}||_F = \eta^2$ due to Lemma 4.

$$||\mathbf{AB}^{(1)}||_F = ||\boldsymbol{Q}\boldsymbol{Q}^T||_F\, \eta^2 \geq ||\boldsymbol{Q}\boldsymbol{Q}^T||_2\, \eta^2 = ||\boldsymbol{Q}||_2^2\, \eta^2 = \eta^2$$

Therefore

$$\max(||d_{\mathcal{S}}(\mathbf{0}) - \mathbf{AB}||_F, ||d_{\mathcal{S}}(\mathbf{0}) + \mathbf{AB}||_F) \geq \eta^2$$

Therefore, there exists matrices $\mathbf{A}$ and ($\mathbf{B}^{(1)}$ or $\mathbf{B}^{(-1)}$) such that given $\eta = 1$, the decoding error is ≥ 1, when recovery threshold is set to $m-1$. $\square$

Lemma 4. *Choose* $\bar{\mathbf{A}} = \boldsymbol{x}\boldsymbol{y}^T$ *and* $\bar{\mathbf{B}} = \boldsymbol{y}\boldsymbol{z}^T$*, then* $||\bar{\mathbf{A}}\bar{\mathbf{B}}||_F = ||\bar{\mathbf{A}}||_F ||\bar{\mathbf{B}}||_F$.

Proof.

$$||\bar{\mathbf{A}}\bar{\mathbf{B}}||_F = ||\boldsymbol{y}||^2 ||\boldsymbol{x}\boldsymbol{z}^T||_F = ||\boldsymbol{y}||^2 \sqrt{\mathrm{Tr}\,(\boldsymbol{z}\boldsymbol{x}^T\boldsymbol{x}\boldsymbol{z}^T)} = ||\boldsymbol{y}||^2 ||\boldsymbol{x}|| ||\boldsymbol{z}||$$
$$||\bar{\mathbf{A}}||_F ||\bar{\mathbf{B}}||_F = \sqrt{\mathrm{Tr}\,(\boldsymbol{y}\boldsymbol{x}^T\boldsymbol{x}\boldsymbol{y}^T)}\sqrt{\mathrm{Tr}\,(\boldsymbol{z}\boldsymbol{y}^T\boldsymbol{y}\boldsymbol{z}^T)} = ||\boldsymbol{y}||^2 ||\boldsymbol{x}|| ||\boldsymbol{z}||$$

$\square$

APPENDIX D
PROOF OF THEOREM 5

We first prove the following crucial theorem.

Theorem 8. *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*

$$\begin{bmatrix} \widehat{a}_0 \\ \vdots \\ \widehat{a}_{m-1} \end{bmatrix} \triangleq \mathbf{V}^{-1} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix}, \tag{56}$$

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

$$|\widehat{a}_{m-1} - a_{m-1}| \leq ||\mathbf{a}||_\infty \cdot (k-m)m\delta. \tag{57}$$

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:

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

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:

$$v[i] = \frac{1}{\prod_{\substack{j=1 \\ j \neq i}}^m (x_j - x_i)}. \tag{58}$$

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

$$\mathbf{v} \cdot \mathbf{r} = \sum_{i=1}^m \frac{\sum_{l=1}^{k-m} a_{m-1+l} x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j \neq i}}^m (x_j - x_i)} = \sum_{l=1}^{k-m} a_{m-1+l} \sum_{i=1}^m \frac{x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j \neq i}}^m (x_j - x_i)}. \tag{59}$$

The expression in (59) can be further simplified using the following lemma.

Lemma 5. *[Theorem 3.2 in [43]]*

$$\sum_{i=1}^{m} \frac{x_i^{m-1+l}}{\prod_{\substack{j=1 \\ j\neq i}}^{m}(x_j - x_i)} = h_l(x_1, \dots, x_m), \tag{60}$$

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

$$h_l(x_1, \dots, x_m) = \sum_{d_1+\cdots+d_m=l} x_1^{d_1} \cdot x_2^{d_2} \cdots x_m^{d_m}. \tag{61}$$

Using Lemma 5, (59) 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,

$$\begin{aligned} |\widehat{a}_{m-1} - a_{m-1}| = |\mathbf{v} \cdot \mathbf{r}| &= \left| \sum_{l=1}^{k-m} a_{m-1+l} \cdot h_l(x_1, \cdots, x_m) \right| \\ &\le \sum_{l=1}^{k-m} a_{m-1+l} \cdot \binom{m+l-1}{l} \delta^l \le ||\mathbf{a}||_\infty \sum_{l=1}^{k-m} \binom{m+l-1}{l} \delta^l \\ &\le ||\mathbf{a}||_\infty (k-m) m\delta. \end{aligned} \tag{62}$$

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

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 6. *Assume $||\mathbf{A}||_F \le \eta$ and $||\mathbf{B}||_F \le \eta$. Then, for $p_{\mathbf{C}}(x)$ given in Construction 3, the ∞-norm of $\mathbf{p}_{[v,w]} = \mathsf{vec}(p_{C[v,w]})$ $(v, w \in [n])$ is bounded as:*

$$||\mathbf{p}_{[v,w]}||_\infty \le \eta^2. \tag{63}$$

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:

$$\mathbf{P}_d = \begin{cases} \sum_{j'-k'=j-k} \mathbf{A}_{i,j'}\mathbf{B}_{k',l} + \sum_{j'-k'=j-k-q} \mathbf{A}_{i+1,j'}\mathbf{B}_{k',l}, & \text{for } j-k > 0, \\ \sum_{j'=k'} \mathbf{A}_{i,j'}\mathbf{B}_{k',l}, & \text{for } j-k = 0. \end{cases} \tag{64}$$

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

$$\begin{bmatrix} \mathbf{A}_{j_1} & \cdots & \mathbf{A}_{j_q} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{B}_{k_1} \\ \vdots \\ \mathbf{B}_{k_q} \end{bmatrix}. \tag{65}$$

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

$$\left|\left| \begin{bmatrix} \mathbf{A}_{j_1} & \cdots & \mathbf{A}_{j_q} \end{bmatrix} \right|\right|_2 \le \eta, \quad \left|\left| \begin{bmatrix} \mathbf{B}_{k_1} \\ \vdots \\ \mathbf{B}_{k_q} \end{bmatrix} \right|\right|_2 \le \eta, \tag{66}$$

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

Proof of Theorem 5. 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:

$$\widehat{\mathbf{C}}_{i,l} = \mathbf{v} \cdot \begin{bmatrix} p_{\mathbf{C}}(x_1) \\ \vdots \\ p_{\mathbf{C}}(x_{d_{i,l}}) \end{bmatrix}. \tag{67}$$

By combining Theorem 8 and Lemma 6, we can show that:

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

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

APPENDIX E
PROOF OF THEOREM 6

Proof. Let $\mathbf{E} = \mathbf{I}_{m \times m} - \sum_{i \in \mathcal{S}_p} d_i^{(p)} \boldsymbol{\alpha}^{(i)} \boldsymbol{\beta}^{(i)T}$. $|| \cdot ||$ here represent Frobenius norm.

$$\begin{aligned} ||\mathbf{C} - \widehat{\mathbf{C}}_{\mathcal{S}_p}|| &= || \sum_{i,j} E[i,j] \mathbf{A}_i \mathbf{B}_j || \le \sum_{i,j} |E[i,j]| ||\mathbf{A}_i \mathbf{B}_j|| \le \sum_{i,j} |E[i,j]| ||\mathbf{AB}|| \\ &\le m ||\mathbf{E}|| \cdot ||\mathbf{AB}|| \le m ||\mathbf{E}|| \cdot ||\mathbf{A}|| \cdot ||\mathbf{B}|| = m \sqrt{\ell^{(p)}} \eta^2. \end{aligned}$$

□