Numerically Stable Coded Matrix Computations via Circulant and Rotation Matrix Embeddings

Aditya Ramamoorthy, *Senior Member, IEEE*, and Li Tang

***Abstract*— Polynomial based methods have recently been used in several works for mitigating the effect of stragglers (slow or failed nodes) in distributed matrix computations. For a system with n worker nodes where s can be stragglers, these approaches allow for an optimal recovery threshold, whereby the intended result can be decoded as long as any $(n-s)$ worker nodes complete their tasks. However, they suffer from serious numerical issues owing to the condition number of the corresponding real Vandermonde-structured recovery matrices; this condition number grows exponentially in n. We present a novel approach that leverages the properties of circulant permutation matrices and rotation matrices for coded matrix computation. In addition to having an optimal recovery threshold, we demonstrate an upper bound on the worst-case condition number of our recovery matrices which grows as $\approx O(n^{s+5.5})$; in the practical scenario where s is a constant, this grows polynomially in n. Our schemes leverage the well-behaved conditioning of complex Vandermonde matrices with parameters on the complex unit circle, while still working with computation over the reals. Exhaustive experimental results demonstrate that our proposed method has condition numbers that are orders of magnitude lower than prior work.**

***Index Terms*— Coded computation, Vandermonde matrix, condition number, numerical stability.**

I. Introduction

PRESENT day computing needs necessitate the usage of large computation clusters that regularly process huge amounts of data on a regular basis. In several of the relevant application domains such as machine learning, datasets are often so large that they cannot even be stored in the disk of a single server. Thus, both storage and computational speed limitations require the computation to be spread over several worker nodes. Such large scale clusters also present attendant operational challenges. These clusters (which can be heterogeneous in nature) suffer from the problem of "stragglers", which are defined as slow nodes (node failures are an extreme form of a straggler). The overall speed of a computational job on these clusters is typically dominated by stragglers in the absence of a sophisticated assignment of tasks to the worker nodes. In particular, simply creating multiple copies of a task to protect against worker node failure can be rather wasteful of computational resources.

In recent years, approaches based on coding theory (referred to as "coded computation") have been effectively used for straggler mitigation. Coded computation offers significant benefits for specific classes of problems such as matrix computations. The essential idea is to create redundant tasks so that the desired result can be recovered as long as a certain number of worker nodes complete their tasks. For instance, suppose that a designated master node wants to compute $\mathbf{A}^T\mathbf{x}$ where the matrix $\mathbf{A}$ is very large. It can decompose $\mathbf{A}$ into block-columns so that $\mathbf{A} = [\mathbf{A}_0 \ \mathbf{A}_1]$ and assign three worker nodes the tasks of determining $\mathbf{A}_0^T\mathbf{x}$, $\mathbf{A}_1^T\mathbf{x}$ and $(\mathbf{A}_0^T + \mathbf{A}_1^T)\mathbf{x}$ respectively. It is easy to see that even if one worker node fails, there is enough information for the master node to compute the final result [1]. Thus, the core idea is to introduce redundancy within the distributed computation by coding across submatrices of the input matrices $\mathbf{A}$ and $\mathbf{B}$. The worker nodes are assigned computational tasks, such that the master node can decode $\mathbf{A}^T\mathbf{B}$ as long as a certain minimum number of the worker nodes complete their tasks.

There have been several works, that have exploited the correspondence of coded computation with erasure codes (see [2] for a tutorial introduction and relevant references). The matrix computation is embedded into the structure of an underlying erasure code and stragglers are treated as erasures. A scheme is said to have a threshold τ if the master node can decode the intended result (matrix-vector or matrix-matrix multiplication) as long any τ nodes complete their tasks. The work of [3], [4] has investigated the tradeoff between the threshold and the tasks assigned to the worker nodes. We discuss related work in more detail in the upcoming Section III.

In this work we examine coded computation from the perspective of numerical stability. Erasure coding typically works with operations over finite fields. Solving a linear system of equation over a finite field only requires the corresponding system to be full-rank. However, when operating over the real field, a numerically robust solution can only be obtained if the condition number (ratio of maximum to minimum singular value) [5] of the system of the equations is small. It turns out that several of the well-known coded computation schemes that work by polynomial evaluation/interpolation have serious numerical stability issues owing to the high condition num-

Manuscript received June 15, 2020; revised June 9, 2021; accepted December 6, 2021. Date of publication December 21, 2021; date of current version March 17, 2022. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1718470 and Grant CCF-1910840. An earlier version of this paper was presented in part at the 2021 IEEE International Symposium on Information Theory (ISIT) [DOI: 10.1109/ISIT45174.2021.9517750]. *(Corresponding author: Aditya Ramamoorthy.)*

Aditya Ramamoorthy is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: adityar@iastate.edu).

Li Tang was with Iowa State University, Ames, IA 50011 USA. He is now with Pinterest, Inc., San Francisco, CA 94107 USA (e-mail: ltang@pinterest.com).

Communicated by A. Mazumdar, Associate Editor for Coding Theory.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2021.3137266.

Digital Object Identifier 10.1109/TIT.2021.3137266

ber of corresponding real Vandermonde system of equations. In this work, we present a scheme that leverages the properties of structured matrices such as circulant permutation matrices and rotation matrices for coded computation. These matrices have eigenvalues that lie on the complex unit circle. Our scheme allows us to exploit the significantly better behaved conditioning of complex Vandermonde matrices while still working with computation over the reals. We also present exhaustive comparisons with existing work.

This paper is organized as follow. Section II presents the problem formulation and Section III overviews related work and summarizes our contributions. Section IV and V discuss our proposed schemes, while Section VI presents numerical experiments and comparisons with existing approaches. Section VII concludes the paper with a discussion of future work. Several of our proofs appear in the Appendix.

II. Problem Formulation

Consider a scenario where the master node has a large $t \times r$ matrix $\mathbf{A} \in \mathbb{R}^{t \times r}$ and either a $t \times 1$ vector $\mathbf{x} \in \mathbb{R}^{t \times 1}$ or a $t \times w$ matrix $\mathbf{B} \in \mathbb{R}^{t \times w}$. The master node wishes to compute $\mathbf{A}^T\mathbf{x}$ or $\mathbf{A}^T\mathbf{B}$ in a distributed manner over n worker nodes in the matrix-vector and matrix-matrix setting respectively. Towards this end, the master node partitions $\mathbf{A}$ (respectively $\mathbf{B}$) into Δ_A (respectively Δ_B) block-columns. Each worker node is assigned $\delta_A \leq \Delta_A$ and $\delta_B \leq \Delta_B$ linearly encoded block-columns of $\mathbf{A}_0, \ldots, \mathbf{A}_{\Delta_A - 1}$ and $\mathbf{B}_0, \ldots, \mathbf{B}_{\Delta_B - 1}$, so that $\delta_A/\Delta_A \leq \gamma_A$ and $\delta_B/\Delta_B \leq \gamma_B$, where γ_A and γ_B represent the storage fraction constraints for $\mathbf{A}$ and $\mathbf{B}$ respectively.

In the matrix-vector case, the i-th worker is assigned encoded submatrices of $\mathbf{A}$ and the vector $\mathbf{x}$ and computes their inner product. In the matrix-matrix case it computes pairwise products of submatrices assigned to it (either all or some subset thereof). We say that a given scheme has *computation threshold* τ if the master node can decode the intended result as long as *any* τ out of n worker nodes complete their tasks. In this case we say that the scheme is resilient to $s = n - \tau$ stragglers. We say that this threshold is *optimal* if the value of τ is the smallest possible for the given storage capacity constraints.

The overall goal is to (i) design schemes that are resilient to s stragglers (s is a design parameter), while ensuring that the (ii) desired result can be decoded in a efficient manner, and (iii) the decoded result is numerically robust even in the presence of round-off errors and other sources of noise.

An analysis of numerical stability is closely related to the condition number of matrices. Let $||\mathbf{M}||$ denote the maximum singular value of a matrix $\mathbf{M}$ of dimension $l \times l$.

Definition 1 (Condition Number): The condition number of a $l \times l$ matrix $\mathbf{M}$ is defined as $\kappa(\mathbf{M}) = ||\mathbf{M}||||\mathbf{M}^{-1}||$. It is infinite if the minimum singular value of $\mathbf{M}$ is zero.

Consider the system of equations $\mathbf{My} = \mathbf{z}$, where $\mathbf{z}$ is known and $\mathbf{y}$ is to be determined. If $\kappa(\mathbf{M}) \approx 10^b$, then the decoded result loses approximately b digits of precision [5]. In particular, matrices that are ill-conditioned lead to significant numerical problems when solving linear equations.

III. Background, Related Work and Summary of Contributions

A significant amount of prior work [3], [4], [6], [7] has demonstrated interesting and elegant approaches based on embedding the distributed matrix computation into the structure of polynomials. Specifically, the encoding at the master node can be viewed as evaluating certain polynomials at distinct real values. Each worker node gets a particular evaluation. When at least τ workers finish their tasks, the master node can decode the intended result by performing polynomial interpolation. The work of [6] demonstrates that when $\mathbf{A}$ and $\mathbf{B}$ are split column-wise and $\delta_A = \delta_B = 1$, the optimal threshold for matrix multiplication is $\Delta_A\Delta_B$ and that polynomial based approaches (henceforth referred to as polynomial codes) achieve this threshold. Prior work has also considered other ways in which the matrices $\mathbf{A}$ and $\mathbf{B}$ can be partitioned. For instance, they can be partitioned both along rows and columns. The work of [3], [4] has obtained threshold results in those cases as well. The so-called Entangled Polynomial and Mat-Dot codes [3], [4], also use polynomial encodings. The key point is that in all these approaches, polynomial interpolation is required when decoding the required result. We note here that to our best knowledge, the idea of embedding matrix multiplication using polynomial maps goes back much further to Yagle [8] (the motivation there was fast matrix multiplication).

Polynomial interpolation corresponds to solving a real *Vandermonde* system of equations at the master node. In the work of [6], this would require solving a $\Delta_A\Delta_B \times \Delta_A\Delta_B$ Vandermonde system. Unfortunately, it can be shown that the condition number of these matrices grows exponentially in $\Delta_A\Delta_B$ [9]. This is a significant drawback and even for systems with around $\Delta_A\Delta_B \approx 30$, the condition number is so high that the decoded results are essentially useless (see Section VI).

In Section VII of [3], it is remarked that when operating over infinite fields such as the reals, one can embed the computation into finite fields to avoid numerical errors. They advocate encoding and decoding over a large enough finite field of prime order p. However, this method would require "quantizing" real matrices $\mathbf{A}$ and $\mathbf{B}$ so that the entries are integers. We demonstrate that the performance of this method can be catastrophically bad. In particular, for this method to work, the maximum possible absolute value of each entry of the quantized matrices, α should be such that $\alpha^2 t < p$, since each entry in the result corresponds to the inner product of columns of $\mathbf{A}$ and columns of $\mathbf{B}$. This "dynamic range constraint (DRC)" means that the error in the computation depends strongly on the actual matrix entries and the value of t is quite limited. If the DRC is violated, the error in the underlying computation can be catastrophic. Even if the DRC is not violated, the dependence of the error on the entries can make it very bad. We discuss this issue in detail in Section VI.

The issue of numerical stability in the coded computation context has been considered in a few recent works [10]–[18]. The work of [11], [13] presented strategies for distributed matrix-vector multiplication and demonstrated some schemes that empirically have better numerical performance than

polynomial based schemes for some values of n and s. However, both these approaches work only for the matrix-vector problem. Reference [14] presents a random convolutional coding approach that applies for both the matrix-vector and the matrix-matrix multiplications problems. Their work demonstrates a computable upper bound on the worst-case condition number of the decoding matrices by drawing on connections with the asymptotic analysis of large Toeplitz matrices. The work of [16] presents constructions that are based on random linear coding ideas where the encoding coefficients are chosen at random from a continuous distribution. These exhibit better condition number properties.

Reference [15] which considers an alternative approach for polynomial based schemes by working within the basis of orthogonal polynomials is most closely related to our work. It demonstrates an upper bound on the worst-case condition number of the decoding matrices which grows as $O(n^{2s})$ where s is the number of stragglers that the scheme is resilient to. They also demonstrate experimentally that their performance is better than the polynomial code approach. In contrast we demonstrate an upper bound that is $\approx O(n^{s+5.5})$. Furthermore, in Section VI we show that in numerical experiments our worst-case condition numbers are much better than [15] (even when $s \leq 6$).

A. Summary of Contributions

The work of [9] shows that unless all (or almost all) the parameters of the Vandermonde matrix lie on the unit circle, its condition number is badly behaved. However, most of these parameters are complex-valued (except ± 1), whereas our matrices $\mathbf{A}$ and $\mathbf{B}$ are real-valued. Using complex evaluation points in the polynomial code scheme, will increase the cost of computations approximately four times for matrix-matrix multiplication and around two times for matrix-vector multiplication. This is an unacceptable hit in computation time.

The main idea of our work is to consider alternate embeddings of distributed matrix computations that are based on rotation and circulant permutation matrices. We demonstrate that these are significantly better behaved from a numerical stability perspective. Furthermore, the worker nodes only work with real computation, thus our method does not incur the complex arithmetic overhead.

- Our main finding in this paper is that we can work with matrix embeddings that allow the worker nodes to perform real-valued computation. Our scheme (i) continues to have the *optimal* threshold of polynomial based approaches when the storage fractions are $\frac{1}{k_A}$ and $\frac{1}{k_B}$ and (ii) enjoys the low condition number of complex Vandermonde matrices with all parameters on the unit circle. In particular, we demonstrate that rotation matrices and circulant permutation matrices of appropriate sizes can be used within the framework of polynomial codes. At the top level, instead of evaluating polynomials at real values, our approach evaluates the polynomials at matrices.
- Using these embeddings we show that the worst-case condition number over all $\binom{n}{n-s}$ possible recovery matrices is upper bounded by $\approx O(n^{s+5.5})$. Furthermore, our experimental results indicate that the actual values are significantly smaller, i.e., the analytical upper bounds are pessimistic.
- An exhaustive numerical comparison with other approaches in the literature shows that the numerical stability of our scheme is currently the best known.

Table I contains a comparison of our work with other schemes in the literature. The columns indicate the corresponding storage fractions, matrix splitting methods, threshold and bounds on the condition number.

IV. Numerically Stable Distributed Matrix Computation Schemes

Our schemes in this work will be defined by the encoding matrices used by the master node, which are such that the master node only needs to perform scalar multiplications and additions. The computationally intensive tasks, i.e., matrix operations are performed by the worker nodes. We begin by defining certain classes of matrices, discuss their relevant properties and present an example that outlines the basic idea of our work.

In what follows, we let $\mathrm{i} = \sqrt{-1}$ and let $[m]$ denote the set $\{0, \ldots, m-1\}$. For a matrix $\mathbf{M}$, $\mathbf{M}(i,j)$ denotes its (i,j)-th entry, whereas $\mathbf{M}_{i,j}$ denotes the (i,j)-th block sub-matrix of $\mathbf{M}$. We use MATLAB inspired notation at certain places. For instance, $\mathrm{diag}(a_1, a_2, \ldots, a_m)$ denotes a $m \times m$ diagonal matrix with a_i's on the diagonal and $\mathbf{M}(:,j)$ denotes the j-th column of matrix $\mathbf{M}$. The notation $\mathbf{M}_1 \otimes \mathbf{M}_2$ denotes the Kronecker product of $\mathbf{M}_1$ and $\mathbf{M}_2$ and the superscript $*$ for a matrix denotes the complex conjugation operator.

Definition 2 (Rotation Matrix): The 2×2 matrix $\mathbf{R}_\theta$ below is called a rotation matrix.

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \mathbf{Q}\Lambda\mathbf{Q}^*, \text{ where} \tag{1}$$

$$\mathbf{Q} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathrm{i} & -\mathrm{i} \\ 1 & 1 \end{bmatrix}, \text{ and } \Lambda = \begin{bmatrix} e^{\mathrm{i}\theta} & 0 \\ 0 & e^{-\mathrm{i}\theta} \end{bmatrix}. \tag{2}$$

Definition 3 (Circulant Permutation Matrix): Let $\mathbf{e}$ be a row vector of length m with $\mathbf{e} = [0\ 1\ 0\ \ldots\ 0]$. Let $\mathbf{P}$ be a $m \times m$ matrix with $\mathbf{e}$ as its first row. The remaining rows are obtained by cyclicly shifting the first row with the shift index equal to the row index. Then $\mathbf{P}^i, i \in [m]$ are said to be circulant permutation matrices. Let $\mathbf{W}$ denote the m-point Discrete Fourier Transform (DFT) matrix, i.e., $\mathbf{W}(i,j) = \frac{1}{\sqrt{m}}\omega_m^{ij}$ for $i \in [m], j \in [m]$ where $\omega_m = e^{\mathrm{i}\frac{2\pi}{\mathrm{m}}}$ denotes the m-th root of unity. Then, it can be shown [19] that $\mathbf{P} = \mathbf{W}\mathrm{diag}(1, \omega_m, \omega_m^2, \ldots, \omega_m^{(m-1)})\mathbf{W}^*$.

Example 1: For $m = 4$, the four possible circulation permutation matrices are

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P}^0 = \mathbf{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{P}^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \mathbf{P}^3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

TABLE I

COMPARISON WITH EXISTING SCHEMES IN THE LITERATURE. THE LAST COLUMN INDICATES THE KNOWN ANALYTICAL RESULTS ABOUT THE WORST-CASE CONDITION NUMBER OF THE CORRESPONDING RECOVERY MATRICES. THE ABBREVIATIONS M-V AND M-M IN THE LAST FOUR ROWS REFER TO MATRIX-VECTOR AND MATRIX-MATRIX MULTIPLICATION, RESPECTIVELY. FOR THE M-V CASES ONLY THE STORAGE FRACTION γ_A IS RELEVANT. FOR THE CIRCULANT EMBEDDING $\tilde{q}$ NEEDS TO BE PRIME. THE CONSTANT $c_1 = 5.5$

CODE	STORAGE FRACTION (γ_A, γ_B)	MATRIX SPLIT	THRESHOLD (τ)	CONDITION NUMBER
POLYNOMIAL [6]	$1/k_A, 1/k_B$	COLUMN-WISE	$k_A k_B$	$\geq \Omega(e^{\tau})$
ENT. POLYNOMIAL [3]	$1/pk_A, 1/pk_B$	ROW AND COLUMN-WISE	$pk_A k_B + p - 1$	$\geq \Omega(e^{\tau})$
ORTHO-POLY [15]	$1/k_A, 1/k_B$	COLUMN-WISE	$k_A k_B$	$\leq O(n^{2(n-\tau)})$
ORTHO-POLY [15]	$1/pk_A, 1/pk_B$	ROW AND COLUMN-WISE	$4k_A k_B p - 2(k_A k_B + pk_A + pk_B) + k_A + k_B + 2p - 1$	$\leq O(n^{2(n-\tau)})$
CONVOL. [14]	$1/k_A, 1/k_B$	COLUMN-WISE	$k_A k_B$	COMPUTABLE UPPER BOUND
RKRP [16]	$1/k_A, 1/k_B$	COLUMN-WISE	$k_A k_B$	ANALYTICAL UPPER BOUND UNKNOWN
ROT. EMBED. (M-V)	$1/k_A$	COLUMN-WISE	k_A	$O(n^{n-\tau+c_1})$
CIRC EMBED. (M-V)	$\tilde{q}/k_A(\tilde{q}-1)$	COLUMN-WISE	k_A	$O(n^{n-\tau+c_1})$
ROT. EMBED. (M-M)	$1/k_A, 1/k_B$	COLUMN-WISE	$k_A k_B$	$\leq O(n^{n-\tau+c_1})$
ROT. EMBED. (M-M)	$1/pk_A, 1/pk_B$	ROW AND COLUMN-WISE	$2pk_A k_B - 1$	$\leq O(n^{n-\tau+c_1})$

Remark 1: Rotation matrices and circulant permutation matrices have the useful property that they are "real" matrices with complex eigenvalues that lie on the unit circle. We use this property extensively in the sequel.

Definition 4 (Vandermonde Matrix): A $m \times m$ Vandermonde matrix $\mathbf{V}$ with parameters $s_0, s_1, \ldots, s_{m-1} \in \mathbb{C}$ is such that $\mathbf{V}(i,j) = s_j^i, i \in [m], j \in [m]$. If the s_i's are distinct, then $\mathbf{V}$ is nonsingular [20]. In this work, we will also assume that the s_i's are non-zero.

Condition Number of Vandermonde Matrices: Let $\mathbf{V}$ be a $m \times m$ Vandermonde matrix with parameters $s_0, s_1, \ldots, s_{m-1}$. The following facts about $\kappa(\mathbf{V})$ follow from prior work [9].

- *Real Vandermonde matrices.* If $s_i \in \mathbb{R}, i \in [m]$, i.e., if $\mathbf{V}$ is a real Vandermonde matrix, then it is known that its condition number is exponential in m.
- *Complex Vandermonde matrices with parameters "not" on the unit circle.* Suppose that the s_i's are complex and let $s_+ = \max_{i=0}^{m-1} |s_i|$. If $s_+ > 1$ then $\kappa(\mathbf{V})$ is exponential in m. Furthermore, if $1/|s_i| \geq \nu > 1$ for at least $\beta \leq m$ of the m parameters, then $\kappa(\mathbf{V})$ is exponential in β.

Based on the above facts, the only scenario where the condition number is somewhat well-behaved is if most or all of the parameters of $\mathbf{V}$ are complex and lie on the unit-circle. In Section C in the appendix, we show the following result which is one of our key technical contributions.

Theorem 1: Consider a $m \times m$ Vandermonde matrix $\mathbf{V}$ where $m < q$ (where q is odd) with distinct parameters $\{s_0, s_1, \ldots, s_{m-1}\} \subset \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$. Let $c_1 = 5.5$. Then,

$$\kappa(\mathbf{V}) \leq O(q^{q-m+c_1}).$$

Remark 2: For the remainder of the paper, we continue to use this theorem with $c_1 = 5.5$. If $q - m$ is a constant, then $\kappa(\mathbf{V})$ grows only polynomially in q. In the subsequent discussion, we will leverage Theorem 1 extensively.

Example 2 (Polynomial Codes): Consider the matrix-vector case where $\Delta_A = 3$ and $\delta_A = 1$. In the polynomial approach, the master node forms $\mathbf{A}(z) = \mathbf{A}_0 + \mathbf{A}_1 z + \mathbf{A}_2 z^2$ and evaluates it at distinct real values $z_1, \ldots, z_n$. The i-th evaluation is sent to the i-th worker node which computes $\mathbf{A}^T(z_i)\mathbf{x}$. From polynomial interpolation, it follows that as long as the master node receives results from any three workers, it can decode $\mathbf{A}^T\mathbf{x}$. However, when Δ_A is large, the interpolation is numerically unstable [9].

The basic idea of our approach to tackle the numerical stability issue is as follows. We further split each $\mathbf{A}_i$ into two equal sized block-columns. Thus, we now have six block-columns, indexed as $\mathbf{A}_0, \ldots \mathbf{A}_5$. Consider the 6×2 matrix defined below; its columns are specified by $\mathbf{g}_0$ and $\mathbf{g}_1$.

$$[\mathbf{g}_0 \ \mathbf{g}_1] = \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^i \\ \mathbf{R}_\theta^{2i} \end{bmatrix}$$

The master node forms "two" encoded matrices for the i-th worker: $\sum_{j=0}^{5} \mathbf{A}_j \mathbf{g}_0(j)$ and $\sum_{j=0}^{5} \mathbf{A}_j \mathbf{g}_1(j)$ (where $\mathbf{g}_i(l)$ denotes the l-th component of the vector $\mathbf{g}_i$). Thus, the storage capacity constraint fraction γ_A is still $\frac{1}{3}$.

Worker node i computes the inner product of these two encoded matrices with $\mathbf{x}$ and sends the result to the master node. It turns out that in this case when any three workers i_0, i_1, and i_2 complete their tasks, the decodability and numerical stability of recovering $\mathbf{A}^T\mathbf{x}$ depends on the condition

number of the following matrix.

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{R}_\theta^{i_0} & \mathbf{R}_\theta^{i_1} & \mathbf{R}_\theta^{i_2} \\ \mathbf{R}_\theta^{2i_0} & \mathbf{R}_\theta^{2i_1} & \mathbf{R}_\theta^{2i_2} \end{bmatrix}.$$

Using the eigen-decomposition of $\mathbf{R}_\theta$ (*cf.* (1)) the above block matrix can expressed as

$$\begin{bmatrix} \mathbf{Q} & 0 & 0 \\ 0 & \mathbf{Q} & 0 \\ 0 & 0 & \mathbf{Q} \end{bmatrix} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \Lambda^{i_0} & \Lambda^{i_1} & \Lambda^{i_2} \\ \Lambda^{2i_0} & \Lambda^{2i_1} & \Lambda^{2i_2} \end{bmatrix}}_{\mathbf{\Sigma}} \begin{bmatrix} \mathbf{Q}^* & 0 & 0 \\ 0 & \mathbf{Q}^* & 0 \\ 0 & 0 & \mathbf{Q}^* \end{bmatrix}.$$

As the pre- and post-multiplying matrices are unitary, the condition number of the above matrix only depends on the properties of the middle matrix, denoted by $\mathbf{\Sigma}$. In what follows, we show that upon appropriate column and row permutations, $\mathbf{\Sigma}$ can be shown equivalent to a block diagonal matrix where each of the blocks is a Vandermonde matrix with parameters on the unit circle. Thus, the matrix is invertible if the corresponding parameters are distinct. Furthermore, even though we use real computation, the numerical stability of our scheme depends on Vandermonde matrices with parameters on the unit circle. Theorem 1 shows that the condition number of such matrices is much better behaved.

In the sequel we show that this argument can be significantly generalized and adapted for the case of circulant permutation embeddings. The matrix-matrix case requires the development of more ideas that we also present. In this section we consider (i) the matrix-vector case where the storage fraction $\gamma_A = 1/k_A$ and (ii) the matrix-matrix case where the storage fractions are $\gamma_A = 1/k_A, \gamma_B = 1/k_B$ respectively.

A. Matrix Splitting Scheme

We partition the matrices $\mathbf{A}$ and $\mathbf{B}$ into $\Delta_A = k_A\ell$ and $\Delta_B = k_B\ell$ block-columns respectively. However, we use two indices to refer to their respective constituent block-columns as this simplifies our later presentation. To avoid confusion, we use the subscript $\langle i,j\rangle$ to refer to the corresponding (i,j)-th block-columns. In particular $\mathbf{A}_{\langle i,j\rangle}, i \in [k_A], j \in [\ell]$ and $\mathbf{B}_{\langle i,j\rangle}, i \in [k_B], j \in [\ell]$ refer to the (i,j)-th block-column of $\mathbf{A}$ and $\mathbf{B}$ respectively, such that

$$\begin{aligned} \mathbf{A} &= [\mathbf{A}_{\langle 0,0\rangle} \dots \mathbf{A}_{\langle 0,\ell-1\rangle} | \dots | \mathbf{A}_{\langle k_A-1,0\rangle} \dots \mathbf{A}_{\langle k_A-1,\ell-1\rangle}], \text{ and} \\ \mathbf{B} &= [\mathbf{B}_{\langle 0,0\rangle} \dots \mathbf{B}_{\langle 0,\ell-1\rangle} | \dots | \mathbf{B}_{\langle k_B-1,0\rangle} \dots \mathbf{B}_{\langle k_B-1,\ell-1\rangle}]. \end{aligned} \tag{3}$$

B. Distributed Matrix-Vector Multiplication

In the matrix-vector case, the encoding matrix for $\mathbf{A}$ will be specified by a $k_A\ell \times n\ell$ "generator" matrix $\mathbf{G}$ such that

$$\hat{\mathbf{A}}_{\langle i,j\rangle} = \sum_{\alpha\in[k_A],\beta\in[\ell]} \mathbf{G}(\alpha\ell+\beta, i\ell+j)\mathbf{A}_{\langle\alpha,\beta\rangle} \tag{4}$$

for $i \in [n], j \in [\ell]$. The worker node i stores $\hat{\mathbf{A}}_{\langle i,j\rangle}$ for $j \in [\ell]$ and $\mathbf{x}$, i.e., it stores $\gamma_A = \ell/\Delta_A = 1/k_A$ fraction of matrix $\mathbf{A}$. Furthermore, it computes $\hat{\mathbf{A}}_{\langle i,j\rangle}^T\mathbf{x}$ for $j \in [\ell]$ and transmits them to the master node.

Algorithm 1 Encoding Scheme for Distributed Matrix-Vector Multiplication

Input: Matrix $\mathbf{A}$ and vector $\mathbf{x}$. Storage fraction $\gamma_A = 1/k_A$, positive integer ℓ and encoding matrix $\mathbf{G}$ of dimension $k_A\ell \times n\ell$.
Output: Worker task assignment.

Partition $\mathbf{A}$ into Δ_A block-columns as in (3).
for $i = 0$ **to** $n-1$ **do**
  Worker i is assigned $\hat{\mathbf{A}}_{\langle i,j\rangle} = \sum_{\alpha\in[k_A],\beta\in[\ell]} \mathbf{G}(\alpha\ell+\beta, i\ell+j)\mathbf{A}_{\langle\alpha,\beta\rangle}$, for all $j \in [\ell]$ and the vector $\mathbf{x}$.
end for
Worker i computes $\hat{\mathbf{A}}_{\langle i,j\rangle}^T\mathbf{x}$ for $j \in [\ell]$.

Thus, the master node receives $\hat{\mathbf{A}}_{\langle i,j\rangle}^T\mathbf{x}$ of length r/Δ_A for $j \in [\ell]$ from a certain number of worker nodes and wants to decode $\mathbf{A}^T\mathbf{x}$ of length r. Based on our encoding scheme, this can be done by solving a $\Delta_A \times \Delta_A$ linear system of equations r/Δ_A times. The structure of this linear system is inherited from the encoding matrix $\mathbf{G}$. The precise details of the encoding schemes can be found in Algorithm 1 (an example appears above).

1) Rotation Matrix Embedding: Let q be an odd number such that $q \geq n$, $\theta = 2\pi/q$ and $\ell = 2$ (*cf.* block column decomposition in (3)). We choose the generator matrix such that its (i,j)-th block-submatrix for $i \in [k_A], j \in [n]$ is given by

$$\mathbf{G}_{i,j}^{rot} = \mathbf{R}_\theta^{ji}. \tag{5}$$

Theorem 2: The threshold for the rotation matrix based scheme specified above is k_A. Furthermore, the worst-case condition number of the recovery matrices is upper bounded by $O(q^{q-k_A+c_1})$.

Proof: Suppose that workers indexed by $i_0, \dots, i_{k_A-1}$ complete their tasks. We extract the corresponding block-columns of $\mathbf{G}^{rot}$ to obtain

$$\tilde{\mathbf{G}}^{rot} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{R}_\theta^{i_0} & \mathbf{R}_\theta^{i_1} & \cdots & \mathbf{R}_\theta^{i_{k_A-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{R}_\theta^{i_0(k_A-1)} & \mathbf{R}_\theta^{i_1(k_A-1)} & \cdots & \mathbf{R}_\theta^{i_{k_A-1}(k_A-1)} \end{bmatrix}.$$

We note here that the decoder attempts to recover each entry of $\mathbf{A}_{\langle i,j\rangle}^T\mathbf{x}$ from the results sent by the worker nodes. Thus, we can equivalently analyze the decoding by considering the system of equations as

$$\mathbf{m}\tilde{\mathbf{G}}^{rot} = \mathbf{c},$$

where $\mathbf{m}, \mathbf{c} \in \mathbb{R}^{1\times k_A\ell}$ are row-vectors such that

$$\begin{aligned} \mathbf{m} &= [\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}] \\ &= [\mathbf{m}_{\langle 0,0\rangle}, \cdots, \mathbf{m}_{\langle 0,\ell-1\rangle}, \cdots, \\ &\quad \cdots, \mathbf{m}_{\langle k_A-1,0\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,\ell-1\rangle}], \text{ and} \\ \mathbf{c} &= [\mathbf{c}_{i_0}, \cdots, \mathbf{c}_{i_{k_A-1}}] \\ &= [\mathbf{c}_{\langle i_0,0\rangle}, \cdots, \mathbf{c}_{\langle i_0,\ell-1\rangle}, \cdots, \\ &\quad \cdots, \mathbf{c}_{\langle i_{k_A-1},0\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},\ell-1\rangle}]. \end{aligned}$$

In the expression above, terms of the form $\mathbf{m}_{\langle i,j\rangle}$ and $\mathbf{c}_{\langle i,j\rangle}$ are scalars. We need to analyze $\kappa(\tilde{\mathbf{G}}^{rot})$. Towards this end, using the eigenvalue decomposition of $\mathbf{R}_\theta$, we have

$$\tilde{\mathbf{G}}^{rot} = \begin{bmatrix} \mathbf{Q} & & \\ & \ddots & \\ & & \mathbf{Q} \end{bmatrix} \tilde{\mathbf{\Lambda}} \begin{bmatrix} \mathbf{Q}^* & & \\ & \ddots & \\ & & \mathbf{Q}^* \end{bmatrix}, \text{ where} \quad (6)$$

$$\tilde{\mathbf{\Lambda}} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \Lambda^{i_0} & \Lambda^{i_1} & \cdots & \Lambda^{i_{k_A-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \Lambda^{i_0(k_A-1)} & \Lambda^{i_1(k_A-1)} & \cdots & \Lambda^{i_{k_A-1}(k_A-1)} \end{bmatrix}$$

and Λ is specified in (2). Note that the pre- and post-multiplying matrices in the RHS of (6) above are both unitary. Therefore $\kappa(\tilde{\mathbf{G}}^{rot})$ is the same as $\kappa(\tilde{\mathbf{\Lambda}})$ [20].

Using Claim 2 in Section E in the appendix, we can permute $\tilde{\mathbf{\Lambda}}$ to put it in block-diagonal form so that

$$\tilde{\mathbf{\Lambda}}_d = \begin{bmatrix} \tilde{\mathbf{\Lambda}}_d[0] & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{\Lambda}}_d[1] \end{bmatrix},$$

where $\tilde{\mathbf{\Lambda}}_d[0]$ and $\tilde{\mathbf{\Lambda}}_d[1]$ are Vandermonde matrices with parameter sets $\{e^{\mathrm{i}\theta i_0}, \ldots, e^{\mathrm{i}\theta i_{k_A-1}}\}$ and $\{e^{-\mathrm{i}\theta i_0}, \ldots, e^{-\mathrm{i}\theta i_{k_A-1}}\}$ respectively. Note that these parameters are distinct points on the unit circle. Thus, $\tilde{\mathbf{\Lambda}}_d[0]$ and $\tilde{\mathbf{\Lambda}}_d[1]$ are both invertible which implies that $\tilde{\Lambda}$ is invertible. This allows us to conclude that the threshold of the scheme is k_A. The upper bound on the condition number follows from Theorem 1. □

Complexity Analysis: Creating an encoded matrix requires a total of Δ_A scalar multiplications and $\Delta_A - 1$ additions of block-columns of size $t \times r/\Delta_A$. Therefore, the total encoding complexity is given by $O(rtn)$. Each worker node computes the product of submatrix of size $r/\Delta_A \times t$ with a vector of size t, i.e., the computational cost is $O(rt/\Delta_A)$. Finally, the decoding process involves inverting a $\Delta_A \times \Delta_A$ matrix once and using the inverse to solve r/Δ_A systems of equations. Thus, the overall decoding complexity is $O(\Delta_A^3 + r\Delta_A)$ where typically, $r \gg \Delta_A^2$.

2) Circulant Permutation Embedding: Let $\tilde{q}$ be a *prime* number which is greater than or equal to n. We set $\ell = \tilde{q} - 1$, so that $\mathbf{A}$ is sub-divided into $k_A(\tilde{q}-1)$ block-columns as in (3). In this embedding we have an additional step. Specifically, the master node generates the following "precoded" matrices.

$$\mathbf{A}_{\langle i,\tilde{q}-1\rangle} = -\sum_{j=0}^{\tilde{q}-2} \mathbf{A}_{\langle i,j\rangle}, i \in [k_A]. \quad (7)$$

In the subsequent discussion, we work with the set of block-columns $\mathbf{A}_{\langle i,j\rangle}$ for $i \in [k_A], j \in [\tilde{q}]$. The coded submatrices $\hat{\mathbf{A}}_{\langle i,j\rangle}$ for $i \in [n], j \in [\tilde{q}]$ are generated by means of a $k_A\tilde{q} \times n\tilde{q}$ matrix $\mathbf{G}^{circ}$ using Algorithm 1. The (i,j)-th block of $\mathbf{G}^{circ}$ can be expressed as

$$\mathbf{G}^{circ}_{i,j} = \mathbf{P}^{ji}, \text{ for } i \in [k_A], j \in [n], \quad (8)$$

where the matrix $\mathbf{P}$ denotes the $\tilde{q} \times \tilde{q}$ circulant permutation matrix introduced in Definition 3. For this scheme the storage fraction $\gamma_A = \tilde{q}/(k_A(\tilde{q}-1))$, i.e., it is slightly higher than $1/k_A$.

Algorithm 2 Decoding Algorithm for Circulant Permutation Scheme

Input: $\mathbf{G}^{circ}_{\mathcal{I}}$ where $|\mathcal{I}| = k_A$ (block-columns of G corresponding to block-columns in $\mathcal{I}$). Row vector $\mathbf{c}$ corresponding to observed values in one system of equations. Permutation π specified in the proof of Theorem 3.

Output: $\mathbf{m}$ which is the solution to $\mathbf{m}\mathbf{G}^{circ}_{\mathcal{I}} = \mathbf{c}$.

1. procedure: Block Fourier Transform and permute $\mathbf{c}$.

for $j = 0$ **to** $k_A - 1$ **do**

Apply FFT to $\mathbf{c}_{i_j} = [\mathbf{c}_{\langle i_j,0\rangle}, \cdots, \mathbf{c}_{\langle i_j,\tilde{q}-1\rangle}]$ to obtain $\mathbf{c}^{\mathcal{F}}_{i_j} = [\mathbf{c}^{\mathcal{F}}_{\langle i_j,0\rangle}, \cdots, \mathbf{c}^{\mathcal{F}}_{\langle i_j,\tilde{q}-1\rangle}]$.

end for

Permute $\mathbf{c}^{\mathcal{F}} = [\mathbf{c}^{\mathcal{F}}_{i_0}, \cdots, \mathbf{c}^{\mathcal{F}}_{i_{k_A-1}}]$ by π to obtain $\mathbf{c}^{\mathcal{F},\pi} = [\mathbf{c}^{\mathcal{F},\pi}_0, \cdots, \mathbf{c}^{\mathcal{F},\pi}_{\tilde{q}-1}]$ where $\mathbf{c}^{\mathcal{F},\pi}_j = [\mathbf{c}^{\mathcal{F}}_{\langle i_0,j\rangle}, \mathbf{c}^{\mathcal{F}}_{\langle i_1,j\rangle}, \cdots, \mathbf{c}^{\mathcal{F}}_{\langle i_{k_A-1},j\rangle}]$, for $j = 0, \ldots, \tilde{q}-1$.

end procedure

2. procedure: Decode $\mathbf{m}^{\mathcal{F},\pi}$ from $\mathbf{c}^{\mathcal{F},\pi}$.

For $i \in \{1, \ldots, \tilde{q}-1\}$, decode $\mathbf{m}^{\mathcal{F},\pi}_i$ from $\mathbf{c}^{\mathcal{F},\pi}_i$ by polynomial interpolation or matrix inversion of $\tilde{\mathbf{G}}^{\mathcal{F}}_d[i]$ (see (13) in Section B in the appendix). Set $\mathbf{m}^{\mathcal{F},\pi}_0 = [0, \cdots, 0]$.

end procedure

3. procedure: Inverse permute and Block Inverse Fourier Transform $\mathbf{m}^{\mathcal{F},\pi}$.

Permute $\mathbf{m}^{\mathcal{F},\pi}$ by π^{-1} to obtain $\mathbf{m}^{\mathcal{F}} = [\mathbf{m}^{\mathcal{F}}_0, \cdots, \mathbf{m}^{\mathcal{F}}_{k_A-1}]$. Apply inverse FFT to each $\mathbf{m}^{\mathcal{F}}_i$ in $\mathbf{m}^{\mathcal{F}}$ to obtain $\mathbf{m} = [\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}]$.

end procedure

Theorem 3: The threshold for the circulant permutation based scheme specified above is k_A. Furthermore, the worst-case condition number of the recovery matrices is upper bounded by $O(\tilde{q}^{\tilde{q}-k_A+c_1})$ and the scheme can be decoded by using Algorithm 2.

The proof appears in Section B in the appendix. It is conceptually similar to the proof of Theorem 2 and relies critically on the fact that all eigenvalues of $\mathbf{P}$ lie on the unit circle and that $\mathbf{P}$ can be diagonalized by the DFT matrix $\mathbf{W}$.

Complexity Analysis: The complexity analysis closely mirrors the analysis for the case of the rotation matrix embedding. However, we note that for the circulant permutation embedding, the $\hat{\mathbf{A}}_{\langle i,j\rangle}$'s can simply be generated by additions since $\mathbf{G}^{circ}$ is a binary matrix. Furthermore, the fact that $\mathbf{P}$ can be diagonalized by the DFT matrix $\mathbf{W}$ suggests an efficient decoding algorithm where the fast Fourier Transform (FFT) plays a key role (see Algorithm 2). In particular, we have the following claim (see Section A in the appendix for proof).

Claim 1: The decoding complexity of recovering $\mathbf{A}^T\mathbf{x}$ is $O(r(\log \tilde{q} + \log^2 k_A))$.

Remark 3: Both circulant permutation matrices and rotation matrices allow us to achieve a specified threshold for distributed matrix vector multiplication. The required storage fraction γ_A is slightly higher for the circulant permutation case and it requires $\tilde{q}$ to be prime. However, it allows for an

efficient FFT based decoding algorithm. On the other hand, the rotation matrix case requires a smaller Δ_A, but the decoding requires solving the corresponding system of equations the complexity of which can be cubic in Δ_A. We note that when the size of $\mathbf{A}$ is large, the decoding time will be much lesser than the worker node computation time; we demonstrate this numerically as well in Section VI. In Section VI, we show results that demonstrate that the normalized mean-square error when circulant permutation matrices are used is lower than the rotation matrix case.

C. Distributed Matrix-Matrix Multiplication

The matrix-matrix case requires the introduction of newer ideas within this overall framework. In this case, a given worker obtains encoded block-columns of both $\mathbf{A}$ and $\mathbf{B}$ and representing the underlying computations is somewhat more involved. Once again we let $\theta = 2\pi/q$, where $q \geq n$ (n is the number of worker nodes) is an odd integer and set $\ell = 2$. Furthermore, let $k_A k_B < n$. The (i,j)-th blocks of the encoding matrices are given by appropriate powers of rotation matrices, i.e.,

$$\mathbf{G}^A_{i,j} = \mathbf{R}_\theta^{ji}, \text{ for } i \in [k_A], j \in [n], \text{ and}$$
$$\mathbf{G}^B_{i,j} = \mathbf{R}_\theta^{(jk_A)i}, \text{ for } i \in [k_B], j \in [n].$$

The master node operates according to the encoding rule discussed previously in the matrix-vector case; the details can be found in Algorithm 3. Thus, each worker node stores $\gamma_A = 1/k_A$ and $\gamma_B = 1/k_B$ fraction of $\mathbf{A}$ and $\mathbf{B}$ respectively. The i-th worker node computes the pair-wise product of the matrices $\hat{\mathbf{A}}^T_{\langle i,l_1\rangle}\hat{\mathbf{B}}_{\langle i,l_2\rangle}$ for $l_1, l_2 = 0, 1$ and returns the result to the master node. Thus, the master node needs to recover all pair-wise products of the form $\mathbf{A}^T_{\langle i,\alpha\rangle}\mathbf{B}_{\langle j,\beta\rangle}$ for $i \in [k_A], j \in [k_B]$ and $\alpha, \beta = 0, 1$. Let $\mathbf{Z}$ denote a $1 \times 4k_Ak_B$ block matrix that contains all of these pair-wise products. The details of the encoding scheme can be found in Algorithm 3 (an example appears below).

Example 3: Suppose $k_A = 2$, $k_B = 2$. Let $n = q = 5$, $\theta = 2\pi/5$. The matrix $\mathbf{A}$ and $\mathbf{B}$ can be partitioned as follows.

$$\mathbf{A} = [\mathbf{A}_{\langle 0,0\rangle} \;\; \mathbf{A}_{\langle 0,1\rangle} \mid \mathbf{A}_{\langle 1,0\rangle} \;\; \mathbf{A}_{\langle 1,1\rangle}], \text{ and}$$
$$\mathbf{B} = [\mathbf{B}_{\langle 0,0\rangle} \;\; \mathbf{B}_{\langle 0,1\rangle} \mid \mathbf{B}_{\langle 1,0\rangle} \;\; \mathbf{B}_{\langle 1,1\rangle}].$$

The encoding matrices $\mathbf{G}^A$ and $\mathbf{G}^B$ are given by

$$\mathbf{G}^A = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{R}_\theta & \mathbf{R}_\theta^2 & \mathbf{R}_\theta^3 & \mathbf{R}_\theta^4 \end{bmatrix}, \text{ and}$$
$$\mathbf{G}^B = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{R}_\theta^2 & \mathbf{R}_\theta^4 & \mathbf{R}_\theta^6 & \mathbf{R}_\theta^8 \end{bmatrix}.$$

Thus, for the i-th worker node, the encoded matrices are obtained as

$$\hat{\mathbf{A}}_{\langle i,0\rangle} = \mathbf{A}_{\langle 0,0\rangle} + \mathbf{R}_\theta^i(0,0)\mathbf{A}_{\langle 1,0\rangle} + \mathbf{R}_\theta^i(1,0)\mathbf{A}_{\langle 1,1\rangle},$$
$$\hat{\mathbf{A}}_{\langle i,1\rangle} = \mathbf{A}_{\langle 0,1\rangle} + \mathbf{R}_\theta^i(0,1)\mathbf{A}_{\langle 1,0\rangle} + \mathbf{R}_\theta^i(1,1)\mathbf{A}_{\langle 1,1\rangle},$$
$$\hat{\mathbf{B}}_{\langle i,0\rangle} = \mathbf{B}_{\langle 0,0\rangle} + \mathbf{R}_\theta^{2i}(0,0)\mathbf{B}_{\langle 1,0\rangle} + \mathbf{R}_\theta^{2i}(1,0)\mathbf{B}_{\langle 1,1\rangle}, \text{ and}$$
$$\hat{\mathbf{B}}_{\langle i,1\rangle} = \mathbf{B}_{\langle 0,1\rangle} + \mathbf{R}_\theta^{2i}(0,1)\mathbf{B}_{\langle 1,0\rangle} + \mathbf{R}_\theta^{2i}(1,1)\mathbf{B}_{\langle 1,1\rangle}.$$

Algorithm 3 Encoding Scheme for Distributed Matrix-Matrix Multiplication

Input: Matrices $\mathbf{A}$ and $\mathbf{B}$. Storage fractions $\gamma_A = 1/k_A, \gamma_B = 1/k_B$, positive integer ℓ and encoding matrices $\mathbf{G}^A$ and $\mathbf{G}^B$ of dimensions $k_A\ell \times n\ell$ and $k_B\ell \times n$ respectively.
Output: Worker task assignment.

Partition $\mathbf{A}$ and $\mathbf{B}$ into Δ_A and Δ_B block-columns as in (3).
for $i = 0$ **to** $n-1$ **do**
Worker i is assigned

$$\hat{\mathbf{A}}_{\langle i,j\rangle} = \sum_{\alpha\in[k_A],\beta\in[\ell]} \mathbf{G}^A(\alpha\ell+\beta, i\ell+j)\mathbf{A}_{\langle\alpha,\beta\rangle}, \text{ and}$$
$$\hat{\mathbf{B}}_{\langle i,j\rangle} = \sum_{\alpha\in[k_B],\beta\in[\ell]} \mathbf{G}^B(\alpha\ell+\beta, i\ell+j)\mathbf{B}_{\langle\alpha,\beta\rangle}$$

for all $j \in [\ell]$.
end for
Worker i computes $\hat{\mathbf{A}}^T_{\langle i,l_1\rangle}\hat{\mathbf{B}}_{\langle i,l_2\rangle}$ for all pairs $l_1 \in [\ell], l_2 \in [\ell]$.

The i-th worker node computes $\hat{\mathbf{A}}^T_{\langle i,0\rangle}\hat{\mathbf{B}}_{\langle i,0\rangle}$, $\hat{\mathbf{A}}^T_{\langle i,0\rangle}\hat{\mathbf{B}}_{\langle i,1\rangle}$, $\hat{\mathbf{A}}^T_{\langle i,1\rangle}\hat{\mathbf{B}}_{\langle i,0\rangle}$, $\hat{\mathbf{A}}^T_{\langle i,1\rangle}\hat{\mathbf{B}}_{\langle i,1\rangle}$. We can represent the computations in the i-th worker node using Kronecker products. We take $\hat{\mathbf{A}}^T_{\langle i,0\rangle}\hat{\mathbf{B}}_{\langle i,1\rangle}$ as an example. Let $\mathbf{Z}$ denote a 1×16 block matrix that contains all of the pair-wise products $\mathbf{A}^T_{\langle a,k_1\rangle}\mathbf{B}_{\langle b,k_2\rangle}$, $a, b, k_1, k_2 = 0, 1$. Consider the following vector (of length 16).

$$\begin{bmatrix} \mathbf{I}(0,0) \\ \mathbf{I}(1,0) \\ \mathbf{R}_\theta^i(0,0) \\ \mathbf{R}_\theta^i(1,0) \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I}(0,1) \\ \mathbf{I}(1,1) \\ \mathbf{R}_\theta^{2i}(0,1) \\ \mathbf{R}_\theta^{2i}(1,1) \end{bmatrix}.$$

Then the computation of $\hat{\mathbf{A}}^T_{\langle i,0\rangle}\hat{\mathbf{B}}_{\langle i,1\rangle}$ can be denoted as the product of each of the elements of $\mathbf{Z}$ with the corresponding component of the above vector followed by their sum. For the sake of convenience we represent this operation by the $\cdot$ operator below. Then we can verify that the computations in i-th worker node can be denoted as

$$\mathbf{Z} \cdot \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^i \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{2i} \end{bmatrix}.$$

Suppose that four different worker nodes i_0, i_1, i_2, i_3 have finished their work. The master node obtains

$$\mathbf{Z} \cdot \mathbf{G}_d = \mathbf{Z} \cdot \Bigg(\begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{i_0} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{2i_0} \end{bmatrix} \Bigg| \Bigg| \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{i_1} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{2i_1} \end{bmatrix} \Bigg| \Bigg| \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{i_2} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{2i_2} \end{bmatrix} \Bigg| \Bigg| \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{i_3} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^{2i_3} \end{bmatrix} \Bigg).$$

We formalize the above construction and prove the $\mathbf{G}_d$ has full rank in Theorem 4.

Theorem 4: The threshold for the rotation matrix based matrix-matrix multiplication scheme is k_Ak_B. The worst-case condition number is bounded by $O(q^{q-k_Ak_B+c_1})$.

Proof: Let $\tau = k_Ak_B$ and suppose that the workers indexed by $i_0, \ldots, i_{\tau-1}$ complete their tasks. Let $\mathbf{G}^A_l$ denote

the l-th block column of $\mathbf{G}^A$ (with similar notation for $\mathbf{G}^B$). Note that for $k_1, k_2 \in \{0,1\}$ the l-th worker node computes $\hat{\mathbf{A}}^T_{\langle l,k_1\rangle}\hat{\mathbf{B}}_{\langle l,k_2\rangle}$ which can be written as

$$\left(\sum_{\alpha\in[k_A],\beta\in\{0,1\}} \mathbf{G}^A(2\alpha+\beta,2l+k_1)\mathbf{A}^T_{\langle\alpha,\beta\rangle}\right)\times \left(\sum_{\alpha\in[k_B],\beta\in\{0,1\}} \mathbf{G}^B(2\alpha+\beta,2l+k_2)\mathbf{B}_{\langle\alpha,\beta\rangle}\right) \equiv \mathbf{Z}\cdot(\mathbf{G}^A(:,2l+k_1)\otimes\mathbf{G}^B(:,2l+k_2)),$$

using the properties of the Kronecker product. Based on this, it can be observed that the decodability of $\mathbf{Z}$ at the master node is equivalent to checking whether the following matrix is full-rank.

$$\tilde{\mathbf{G}} = [\mathbf{G}^A_{i_0}\otimes\mathbf{G}^B_{i_0}|\mathbf{G}^A_{i_1}\otimes\mathbf{G}^B_{i_1}|\ldots|\mathbf{G}^A_{i_{\tau-1}}\otimes\mathbf{G}^B_{i_{\tau-1}}].$$

To analyze this matrix, consider the following decomposition of $\mathbf{G}^A_l\otimes\mathbf{G}^B_l$, for $l\in[n]$.

$$\begin{aligned}
&\mathbf{G}^A_l\otimes\mathbf{G}^B_l\\
&=\begin{bmatrix}\mathbf{Q}\mathbf{Q}^*\\ \mathbf{Q}\Lambda^l\mathbf{Q}^*\\ \vdots\\ \mathbf{Q}\Lambda^{l(k_A-1)}\mathbf{Q}^*\end{bmatrix}\otimes\begin{bmatrix}\mathbf{Q}\mathbf{Q}^*\\ \mathbf{Q}\Lambda^{lk_A}\mathbf{Q}^*\\ \vdots\\ \mathbf{Q}\Lambda^{lk_A(k_B-1)}\mathbf{Q}^*\end{bmatrix}\\
&=\left((\mathbf{I}_{k_A}\otimes\mathbf{Q})\begin{bmatrix}\mathbf{I}\\ \Lambda^l\\ \vdots\\ \Lambda^{l(k_A-1)}\end{bmatrix}\begin{bmatrix}\mathbf{Q}^*\end{bmatrix}\right)\otimes\\
&\left((\mathbf{I}_{k_B}\otimes\mathbf{Q})\begin{bmatrix}\mathbf{I}\\ \Lambda^{lk_A}\\ \vdots\\ \Lambda^{lk_A(k_B-1)}\end{bmatrix}\begin{bmatrix}\mathbf{Q}^*\end{bmatrix}\right),
\end{aligned}$$

where the first equality uses the eigen-decomposition of $\mathbf{R}_\theta$. Applying the properties of Kronecker products, this can be simplified as

$$\underbrace{((\mathbf{I}_{k_A}\otimes\mathbf{Q})\otimes(\mathbf{I}_{k_B}\otimes\mathbf{Q}))}_{\tilde{\mathbf{Q}}_1}\times \underbrace{\left(\begin{bmatrix}\mathbf{I}\\ \Lambda^l\\ \vdots\\ \Lambda^{l(k_A-1)}\end{bmatrix}\otimes\begin{bmatrix}\mathbf{I}\\ \Lambda^{lk_A}\\ \vdots\\ \Lambda^{lk_A(k_B-1)}\end{bmatrix}\right)}_{\mathbf{X}_l}\underbrace{\left(\begin{bmatrix}\mathbf{Q}^*\end{bmatrix}^{\otimes 2}\right)}_{\tilde{\mathbf{Q}}_2}.$$

Therefore, we can express

$$\begin{aligned}
\tilde{\mathbf{G}} &= [\mathbf{G}^A_{i_0}\otimes\mathbf{G}^B_{i_0}|\mathbf{G}^A_{i_1}\otimes\mathbf{G}^B_{i_1}|\ldots|\mathbf{G}^A_{i_{\tau-1}}\otimes\mathbf{G}^B_{i_{\tau-1}}]\\
&=\tilde{\mathbf{Q}}_1[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]\begin{bmatrix}\tilde{\mathbf{Q}}_2 & 0 & \ldots & 0\\ 0 & \tilde{\mathbf{Q}}_2 & \ldots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \ldots & \tilde{\mathbf{Q}}_2\end{bmatrix}.
\end{aligned}$$

Once again, we can conclude that the invertibility and the condition number of $\tilde{\mathbf{G}}$ only depends on $[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]$ as the matrices pre- and post- multiplying it are both unitary. The invertibility of $[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]$ follows from an application of Claim 3 in Section E in the appendix. The proof of Claim 3 also shows that upon appropriate row-column permutations, the matrix $[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]$ can be expressed as a block-diagonal matrix with four blocks each of size $\tau\times\tau$. Each of these blocks is a Vandermonde matrix with parameters from the set $\{1,\omega_q,\omega_q^2,\ldots,\omega_q^{q-1}\}$. Therefore, $[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]$ is non-singular and it follows that the threshold of our scheme is k_Ak_B. An application of Theorem 1 implies that the worst-case condition number is at most $O(q^{q-\tau+c_1})$. □

Remark 4: The proofs of Theorem 2 and 4 involve a diagonalization argument with pre- and post-multiplying matrices that are unitary. We emphasize that this is only for the analysis of the scheme and the encoding and decoding schemes do not require multiplication by these matrices.

Complexity Analysis: Creating the $\hat{\mathbf{A}}_{\langle i,l\rangle}$ matrix requires a total of Δ_A scalar multiplications and Δ_A-1 additions of block-columns of size $t\times r/\Delta_A$; a similar argument applies for creating the $\hat{\mathbf{B}}_{\langle i,l\rangle}$ matrix (note that $\Delta_A=2\ k_A, \Delta_B=2k_B$). Thus, the total encoding complexity is given by $O((r+w)tn)$. Each worker node computes four submatrix products. Thus, the worker node computational cost is $O(4\times rtw/\Delta_A\Delta_B)=O(rtw/k_Ak_B)$. The decoding process involves inverting a matrix of dimension $\Delta_A\Delta_B\times\Delta_A\Delta_B$ followed by solving $rw/\Delta_A\Delta_B$ systems of equations. Thus, the overall decoding complexity is given by $O((\Delta_A\Delta_B)^3+rw\Delta_A\Delta_B)$. It can be seen that the decoding complexity is independent of t. Thus, when the input matrices are large, i.e., r, w and t are large, then the overall cost is dominated by the worker node computation time.

V. Generalized Distributed Matrix Multiplication

In the previous section, we consider the case that $\mathbf{A}$ and $\mathbf{B}$ are partitioned into block-columns. In this section, we consider a more general scenario where $\mathbf{A}$ and $\mathbf{B}$ are partitioned into block-columns and block-rows. This construction resembles the entangled polynomial codes of [3].

A. Matrix Splitting Scheme

We partition the matrices $\mathbf{A}$ and $\mathbf{B}$ into $2p$ block-rows and $\Delta_A=k_A$ block-columns, and $2p$ block-rows and $\Delta_B=k_B$ block-columns respectively. We use two indices for the block-rows to simplify our presentation. In particular, we denote

$$\begin{aligned}
&\mathbf{A}=[\mathbf{A}_{(\langle i,l\rangle,j)}], i\in[p], l\in\{0,1\}, j\in[k_A], \text{ and}\\
&\mathbf{B}=[\mathbf{B}_{(\langle i,l\rangle,j)}], i\in[p], l\in\{0,1\}, j\in[k_B],
\end{aligned}\tag{9}$$

where $\mathbf{A}_{(\langle i,l\rangle,j)}$ denotes the submatrix indexed by the $\langle i,l\rangle$-th block row and j-th block-column of $\mathbf{A}$. A similar interpretation holds for $\mathbf{B}_{(\langle i,l\rangle,j)}$. We let $\theta=2\pi/q$, where $q\geq n>2k_Ak_Bp-1$ (recall that n is the number of worker nodes) is an odd integer.

The encoding in this scenario is more complicated to express. We simplify this by leveraging the following simple result which can be easily verified.

Algorithm 4 Encoding Scheme for Generalized Distributed Matrix-Matrix Multiplication

Input: Matrices $\mathbf{A}$ and $\mathbf{B}$. Storage fractions $\gamma_A = 1/pk_A, \gamma_B = 1/pk_B$. Integer $\zeta = \frac{t}{2p}$.
Output: Worker task assignment.

Partition $\mathbf{A}$ and $\mathbf{B}$ into $2p \times \Delta_A$ and $2p \times \Delta_B$ blocks as in (9).
for $k = 0$ **to** $n-1$ **do**
Worker k is assigned

$$\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}(\mathbf{R}_{-\theta}^{k((j-1)p+i+1)}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{(\langle i,0\rangle,j)}\\ \mathbf{A}_{(\langle i,1\rangle,j)}\end{bmatrix},$$

$$\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1}(\mathbf{R}_{\theta}^{k(p-1-i+jpk_A)}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{(\langle i,0\rangle,j)}\\ \mathbf{B}_{(\langle i,1\rangle,j)}\end{bmatrix}.$$

end for
Worker k computes

$$\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}^T\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}.$$

Lemma 1: Suppose that matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ both have ζ rows and the same column dimension. Consider a 2×2 matrix $\mathbf{\Psi} = [\Psi_{i,j}]$, $i = 0, 1, j = 0, 1$. Then

$$\begin{bmatrix}\Psi_{0,0}\mathbf{M}_1 + \Psi_{0,1}\mathbf{M}_2\\ \Psi_{1,0}\mathbf{M}_1 + \Psi_{1,1}\mathbf{M}_2\end{bmatrix} = (\mathbf{\Psi}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{M}_1\\ \mathbf{M}_2\end{bmatrix}.$$

The complete encoding algorithm appears in Algorithm 4.

The k-th worker node stores $\hat{\mathbf{A}}_{\langle k,l\rangle}$, $\hat{\mathbf{B}}_{\langle k,l\rangle}$, $l = 0, 1$. Thus, each worker node stores $\gamma_A = \frac{2}{2pk_A} = \frac{1}{pk_A}$ and $\gamma_B = \frac{2}{2pk_B} = \frac{1}{pk_B}$ fraction of $\mathbf{A}$ and $\mathbf{B}$ respectively. Worker node k computes

$$\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}^T\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}. \tag{10}$$

Before presenting our decoding algorithm and the main result of this section, we discuss the following example that helps clarify the underlying ideas.

Example 4: Suppose $k_A = 1, k_B = 1, p = 2$. Let $n = 4$. The matrix $\mathbf{A}$ and $\mathbf{B}$ can be partitioned as follows.

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_{(\langle 0,0\rangle,0)}\\ \mathbf{A}_{(\langle 0,1\rangle,0)}\\ \mathbf{A}_{(\langle 1,0\rangle,0)}\\ \mathbf{A}_{(\langle 1,1\rangle,0)}\end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix}\mathbf{B}_{(\langle 0,0\rangle,0)}\\ \mathbf{B}_{(\langle 0,1\rangle,0)}\\ \mathbf{B}_{(\langle 1,0\rangle,0)}\\ \mathbf{B}_{(\langle 1,1\rangle,0)}\end{bmatrix}.$$

In this example, since $k_A = k_B = 1$, there is only one block column in $\mathbf{A}$ and $\mathbf{B}$. Therefore, the index j in $\mathbf{A}_{(\langle i,l\rangle,j)}$ and $\mathbf{B}_{(\langle i,l\rangle,j)}$ is always 0. Accordingly, to simplify our presentation, we only use indices i and l to refer to the respective constituent block rows of $\mathbf{A}$ and $\mathbf{B}$. That is, we simplify $\mathbf{A}_{(\langle i,l\rangle,j)}$ and $\mathbf{B}_{(\langle i,l\rangle,j)}$ to $\mathbf{A}_{\langle i,l\rangle}$ and $\mathbf{B}_{\langle i,l\rangle}$, respectively. Our scheme aims to allow the master node to recover $\mathbf{A}^T\mathbf{B} = \mathbf{A}_{\langle 0,0\rangle}^T\mathbf{B}_{\langle 0,0\rangle} + \mathbf{A}_{\langle 0,1\rangle}^T\mathbf{B}_{(\langle 0,1\rangle} + \mathbf{A}_{\langle 1,0\rangle}^T\mathbf{B}_{\langle 1,0\rangle} + \mathbf{A}_{\langle 1,1\rangle}^T\mathbf{B}_{\langle 1,1\rangle}$. Suppose that $\mathbf{A}_{\langle i,l\rangle}$ and $\mathbf{B}_{\langle i,l\rangle}$ have ζ rows. The encoding process (*cf.* Algorithm 4) can be defined as $\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{1}(\mathbf{R}_{-\theta}^{k(i-1)}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle i,0\rangle}\\ \mathbf{A}_{\langle i,1\rangle}\end{bmatrix}$, and $\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{1}(\mathbf{R}_{\theta}^{k(1-i)}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle i,0\rangle}\\ \mathbf{B}_{\langle i,1\rangle}\end{bmatrix}$.

The computation in worker node k (*cf.* (10)) can be analyzed as follows. Let $\begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{\langle i,0\rangle}\\ \mathbf{A}^{\mathcal{F}}_{\langle i,1\rangle}\end{bmatrix} = (\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle i,0\rangle}\\ \mathbf{A}_{\langle i,1\rangle}\end{bmatrix}$ and $\begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{\langle i,0\rangle}\\ \mathbf{B}^{\mathcal{F}}_{\langle i,1\rangle}\end{bmatrix} = (\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle i,0\rangle}\\ \mathbf{B}_{\langle i,1\rangle}\end{bmatrix}$. Then

$$\begin{aligned}
&\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}^T\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}\\
&\overset{(a)}{=}\left((\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}\right)^*(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}\\
&=\Bigg((\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{R}_{-\theta}^{-k}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 0,0\rangle}\\ \mathbf{A}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{I}_2\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 1,0\rangle}\\ \mathbf{A}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)^*\\
&\quad\Bigg((\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{R}_{\theta}^{k}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 0,0\rangle}\\ \mathbf{B}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{I}_2\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 1,0\rangle}\\ \mathbf{B}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)\\
&\overset{(b)}{=}\Bigg((\mathbf{Q}^*\mathbf{R}_{-\theta}^{-k}\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 0,0\rangle}\\ \mathbf{A}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad(\mathbf{Q}^*\mathbf{I}_2\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 1,0\rangle}\\ \mathbf{A}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)^*\\
&\quad\Bigg((\mathbf{Q}^*\mathbf{R}_{\theta}^{k}\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 0,0\rangle}\\ \mathbf{B}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad(\mathbf{Q}^*\mathbf{I}_2\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 1,0\rangle}\\ \mathbf{B}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)\\
&\overset{(c)}{=}\Bigg(\left(\begin{bmatrix}\omega_q^{*-k} & 0\\ 0 & \omega_q^{*k}\end{bmatrix}\otimes\mathbf{I}_\zeta\right)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 0,0\rangle}\\ \mathbf{A}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\otimes\mathbf{I}_\zeta\right)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 1,0\rangle}\\ \mathbf{A}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)^*\\
&\quad\Bigg(\left(\begin{bmatrix}\omega_q{}^{k} & 0\\ 0 & \omega_q{}^{-k}\end{bmatrix}\otimes\mathbf{I}_\zeta\right)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 0,0\rangle}\\ \mathbf{B}_{\langle 0,1\rangle}\end{bmatrix}+\\
&\quad\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\otimes\mathbf{I}_\zeta\right)(\mathbf{Q}^*\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 1,0\rangle}\\ \mathbf{B}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)\\
&\overset{(d)}{=}\Bigg(\begin{bmatrix}\omega_q^{*-k}\mathbf{A}^{\mathcal{F}}_{\langle 0,0\rangle}\\ \omega_q^{*k}\mathbf{A}^{\mathcal{F}}_{\langle 0,1\rangle}\end{bmatrix}+\begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{\langle 1,0\rangle}\\ \mathbf{A}^{\mathcal{F}}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)^*\\
&\quad\Bigg(\begin{bmatrix}\omega_q{}^{k}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle}\\ \omega_q{}^{-k}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle}\end{bmatrix}+\begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle}\\ \mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle}\end{bmatrix}\Bigg)\\
&=(\mathbf{A}^{\mathcal{F}*}_{\langle 0,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle}+\mathbf{A}^{\mathcal{F}*}_{\langle 1,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle})\omega_q^{-k}+\\
&\quad(\mathbf{A}^{\mathcal{F}*}_{\langle 0,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle}+\mathbf{A}^{\mathcal{F}*}_{\langle 1,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle}+\\
&\quad\mathbf{A}^{\mathcal{F}*}_{\langle 0,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle}+\mathbf{A}^{\mathcal{F}*}_{\langle 1,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle})+\\
&\quad(\mathbf{A}^{\mathcal{F}*}_{\langle 1,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle}+\mathbf{A}^{\mathcal{F}*}_{\langle 0,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle})\omega_q^{k}
\end{aligned}$$

where

- (a) holds because $\mathbf{Q}^*\otimes\mathbf{I}_\zeta$ is unitary,

- (b) holds by the mixed-product property of Kronecker product. For example,

$$\begin{aligned}(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)(\mathbf{R}_{-\theta}^{-k} \otimes \mathbf{I}_\zeta) &= (\mathbf{Q}^* \mathbf{R}_{-\theta}^{-k}) \otimes \mathbf{I}_\zeta \\ &= (\mathbf{Q}^* \mathbf{R}_{-\theta}^{-k} \mathbf{Q}\mathbf{Q}^*) \otimes \mathbf{I}_\zeta \\ &= (\mathbf{Q}^* \mathbf{R}_{-\theta}^{-k} \mathbf{Q} \otimes \mathbf{I}_\zeta)(\mathbf{Q}^* \otimes \mathbf{I}_\zeta).\end{aligned}$$

- (c) holds because $\mathbf{Q}^* \mathbf{R}_\theta \mathbf{Q} = \begin{bmatrix} \omega_q & 0 \\ 0 & \omega_q^{-1} \end{bmatrix}$, and
- (d) holds by Lemma 1.

Thus, it is clear that whenever the master node collects the results of any three distinct worker nodes, it can recover $\mathbf{A}_{\langle 0,0\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle 0,0\rangle}^{\mathcal{F}} + \mathbf{A}_{\langle 1,0\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle 1,0\rangle}^{\mathcal{F}} + \mathbf{A}_{\langle 0,1\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle 0,1\rangle}^{\mathcal{F}} + \mathbf{A}_{\langle 1,1\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle 1,1\rangle}^{\mathcal{F}}$. However, we observe that for $i = 0, 1$

$$\begin{bmatrix} \mathbf{A}_{\langle i,0\rangle}^{\mathcal{F}} \\ \mathbf{A}_{\langle i,1\rangle}^{\mathcal{F}} \end{bmatrix}^* \begin{bmatrix} \mathbf{B}_{\langle i,0\rangle}^{\mathcal{F}} \\ \mathbf{B}_{\langle i,1\rangle}^{\mathcal{F}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\langle i,0\rangle} \\ \mathbf{A}_{\langle i,1\rangle} \end{bmatrix}^T \begin{bmatrix} \mathbf{B}_{\langle i,0\rangle} \\ \mathbf{B}_{\langle i,1\rangle} \end{bmatrix}.$$

Thus, we can equivalently recover $\mathbf{A}^T\mathbf{B}$.

The analysis in the example above can be generalized to show the following result. The proof appears in Section D in the appendix.

Theorem 5: The threshold for scheme in this section is $2pk_Ak_B - 1$. The worst-case condition number of the recovery matrices is upper bounded by $O(q^{q-2pk_Ak_B+1+c_1})$.

Remark 5: When $k_A = k_B = 1$, the threshold of this scheme matches the Entangled Polynomial code [3] and the MatDot codes [4], with the added advantage of excellent numerical stability.

The decoding algorithm in this case requires more steps. It is specified in Algorithm 5. In particular, it requires us to work with the inverse of a complex matrix (see (11)) which is essentially (upto a unitary scaling) a Vandermonde matrix with parameters on the unit circle. The underlying reason can be found by examining the proof of Theorem 5. Thus, the decoding in this case is more expensive than prior methods that work exclusively with real valued decoding. Nevertheless, we emphasize that the worker node computation is still real-valued.

Suppose that the k-th worker node computes $\hat{\mathbf{A}}_k^T \hat{\mathbf{B}}_k$ and that the master node receives the computation results from any $\tau = 2pk_Ak_B - 1$ worker nodes, which are denoted by $i_0, \cdots, i_{\tau-1}$. By (16), the useful and interference terms can be decoded by computing the inverse of

$$\mathbf{G}_{\mathcal{I}}^{vand} = \begin{bmatrix} \omega_q^{-i_0(pk_Ak_B-1)} & \omega_q^{-i_1(pk_Ak_B-1)} & \cdots & \omega_q^{-i_{\tau-1}(pk_Ak_B-1)} \\ \omega_q^{-i_0(pk_Ak_B-2)} & \omega_q^{-i_1(pk_Ak_B-2)} & \cdots & \omega_q^{-i_{\tau-1}(pk_Ak_B-2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ \omega_q^{i_0(pk_Ak_B-1)} & \omega_q^{i_1(pk_Ak_B-1)} & \cdots & \omega_q^{i_{\tau-1}(pk_Ak_B-1)} \end{bmatrix}. \tag{11}$$

and using it to solve $\frac{r}{k_A} \times \frac{w}{k_B}$ systems of equations. We point out that by multiplying $\mathbf{G}_{\mathcal{I}}^{vand}$ from the right by the unitary matrix $[\text{diag}(\omega_q^{i_0}, \ldots, \omega_q^{i_{\tau-1}})]^{pk_Ak_B-1}$, it can be seen that $\kappa(\mathbf{G}_{\mathcal{I}}^{vand})$ is the same as the condition number of a Vandermonde matrix of size $(2pk_Ak_B - 1) \times (2pk_Ak_B - 1)$ with parameters $\omega_q^{i_0}, \ldots, \omega_q^{i_{\tau-1}}$.

Algorithm 5 Decoding Scheme for Generalized Distributed Matrix-Matrix Multiplication

Input: $\mathbf{G}_{\mathcal{I}}^{vand}$ (*cf.* (11)) where $|\mathcal{I}| = 2pk_Ak_B - 1$ (columns of $\mathbf{G}^{vand}$ corresponding to columns in $\mathcal{I}$). Row vectors $\mathbf{c}$ corresponding to the observed values in each of the $\frac{r}{k_A} \times \frac{w}{k_B}$ system of equations.
Output: Decoded estimate $\tilde{\mathbf{C}}$ of $\mathbf{A}^T\mathbf{B}$.

1. procedure: Decode $\hat{\mathbf{m}}$ from $\mathbf{c}$
$\hat{\mathbf{m}} = [\hat{\mathbf{m}}_{-pk_Ak_B}, \cdots, \hat{\mathbf{m}}_0, \cdots, \hat{\mathbf{m}}_{pk_Ak_B}]$ by $\hat{\mathbf{m}} = \mathbf{c}(\mathbf{G}_{\mathcal{I}}^{vand})^{-1}$.
end procedure
2. procedure: Repeat above procedure for each of the $\frac{r}{k_A} \times \frac{w}{k_B}$ systems of equations. Upon appropriate indexing, we can form a matrix $\hat{\mathbf{M}}_{i,j}, -(k_A - 1) \le i \le k_A - 1, -(k_B - 1) \le j \le k_B - 1$ using the decoded components $\hat{\mathbf{m}}_{ip+jpk_A}$.
end procedure
3. procedure: Recover $\tilde{\mathbf{C}}_{i_1,j_1}$ for $i_1 \in [k_A], j_1 \in [k_B]$.
if $i_1 = 0, j_1 = 0$ **then**
$\tilde{\mathbf{C}}_{0,0} = \hat{\mathbf{M}}_{0,0}$.
else
$\tilde{\mathbf{C}}_{i_1,j_1} = \hat{\mathbf{M}}_{i_1,j_1} + \hat{\mathbf{M}}_{-i_1,-j_1}$.
end if
end procedure

Finally, the result $\mathbf{C} = [\mathbf{C}_{i,j}]$, $i \in [k_A]$, $j \in [k_B]$ can be recovered since $\mathbf{C}_{i,j} = \sum_{u=0}^{p-1} (\mathbf{A}_{\langle\langle u,0\rangle,i\rangle}^T \mathbf{B}_{\langle\langle u,0\rangle,j\rangle} + \mathbf{A}_{\langle\langle u,1\rangle,i\rangle}^T \mathbf{B}_{\langle\langle u,1\rangle,j\rangle}) = \left(\sum_{u=0}^{p-1} (\mathbf{A}_{\langle\langle u,0\rangle,i\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle\langle u,0\rangle,j\rangle}^{\mathcal{F}}\right) + \left(\sum_{u=0}^{p-1} \mathbf{A}_{\langle\langle u,1\rangle,i\rangle}^{\mathcal{F}*} \mathbf{B}_{\langle\langle u,1\rangle,j\rangle}^{\mathcal{F}})\right)$. The precise decoding algorithm is summarized in Algorithm 5.

Complexity Analysis: We note here that the decoding algorithm involving inverting a $(2pk_Ak_B - 1) \times (2pk_Ak_B - 1)$ complex Vandermonde matrix once and using the inverse to solve $\frac{r}{k_A} \times \frac{w}{k_B}$ systems of equations in Steps 1 and 2. Step 3 involves the sum of matrices of size $\frac{r}{k_A} \times \frac{w}{k_B}$ so its complexity is $O(rw)$. Thus, the overall decoding complexity is $O((2pk_Ak_B-1)^3 + rw + \frac{rw}{k_Ak_B}(2pk_Ak_B-1)^2) \approx O(p^3k_A^3k_B^3 + rwp^2k_Ak_B)$, where typically, $rw \gg pk_A^2k_B^2$.

VI. Comparisons and Numerical Experiments

We now present a comparison of our techniques with other approaches in the literature. Towards this end we will compare the worst-case and the average condition numbers of the recovery matrices of the different schemes. Furthermore, we will also present corresponding normalized mean-squared-error (MSE) vs. SNR curves. For matrix-vector multiplication, let $\mathbf{A}^T\mathbf{x}$ denote the true value of the computation and $\widehat{\mathbf{A}^T\mathbf{x}}$ denote the result of using one of the discussed methods. The normalized MSE is defined as $\frac{||\mathbf{A}^T\mathbf{x} - \widehat{\mathbf{A}^T\mathbf{x}}||_F}{||\mathbf{A}^T\mathbf{x}||_F}$ (the notation $||\cdot||_F$ denotes the Frobenius norm of the matrix). Similarly, for the matrix-matrix multiplication, the normalized MSE is given by $\frac{||\mathbf{A}^T\mathbf{B} - \widehat{\mathbf{A}^T\mathbf{B}}||_F}{||\mathbf{A}^T\mathbf{B}||_F}$ where $\mathbf{A}^T\mathbf{B}$ is the true product and $\widehat{\mathbf{A}^T\mathbf{B}}$ is

the decoded product using one of the methods. We will also report the computation threshold, worker computation times and decoding times for all the methods under consideration.

Suppose that the number of workers n is odd, so that we can pick $q = n$ for the rotation matrix embedding. From a theoretical perspective our schemes have a worst-case condition number (over the different recovery submatrices) that is upper bounded by $O(q^{q-\tau+c_1})$ where τ is the recovery threshold. Equivalently, the worst-case condition number is upper bounded by $O(n^{s+c_1})$ (recall that $c_1 = 5.5$). We note here that this upper bound is definitely loose and our numerical experiments which will be presented shortly indicate that the actual condition number values are much smaller. The work of [15] shows a condition number upper bound which is $O(n^{2s})$. While this is larger than our upper bound for values of $s \geq 6$ we emphasize that our actual condition number values are much lower than [15] even for $s \leq 6$.

As discussed previously, the scheme of [6] has condition numbers that are exponential in the recovery threshold τ. This is corroborated by our numerical experiments as well. In Section VII of [3], the authors propose a finite field embedding approach as a potential solution to the numerical issues encountered when operating over the reals. For this purpose the real entries will need to multiplied by large enough integers and then quantized so that each entry lies with 0 and $p-1$ for a large enough prime p. All computations will be performed within the finite field of order p, i.e., by reducing the computations modulo-p. This technique requires that each $\mathbf{A}_i^T\mathbf{B}_j$ needs to have all its entries within 0 to $p-1$, otherwise there will be errors in the computation. Let α be an upper bound on the absolute value of matrix entries in $\mathbf{A}$ and $\mathbf{B}$. Then, this means that the following dynamic range constraint (DRC),

$$\alpha^2 t \leq p-1$$

needs to be satisfied. Otherwise, the modulo-p operation will cause arbitrarily large errors.

We note here that the publicly available code for [6] uses $p = 65537$. Now consider a system with $k_A = 3$, $k_B = 2$. Even for small matrices with $\mathbf{A}$ of size 400×200, $\mathbf{B}$ of size 400×300 and entries chosen as random integers between 0 to 30, the DRC is violated for $p = 65537$ since $30^2 \times 400 > 65537$. In this scenario, the normalized MSE of the [6] approach is 0.7746. In contrast, our method has a normalized MSE $\approx 2 \times 10^{-28}$ for the same system with $k_A = 3, k_B = 2$.

When working over 64-bit integers, the largest integer is $\approx 10^{19}$. Thus, even if $t \approx 10^5$, the finite-field embedding method can only support $\alpha \leq 10^7$. Thus, the range is rather limited. Furthermore, considering matrices of limited dynamic range is not a valid assumption. In machine learning scenarios such as deep neural networks, matrix multiplications are applied repeatedly, and the output of one stage serves as the input for the other. Thus, over several iterations the dynamic range of the matrix entries will grow. Consequently, applying this technique will necessarily incur quantization error.

The most serious limitation of the method comes from the fact the error in the computation (owing to quantization) is strongly dependent on the actual entries of the $\mathbf{A}$ and $\mathbf{B}$ matrices. In fact, we can generate structured integer matrices $\mathbf{A}$ and $\mathbf{B}$ such that the normalized MSE of their approach is exactly 1.0. Towards this end we first pick the prime $p = 2147483647$ (which is much larger than their publicly available code) so that their method can support higher dynamic range. Next let $r = w = t = 2000$. This implies that α has to be ≤ 1000 by the dynamic range constraint. For $k_A = k_B = 2$, the matrices have the following block decomposition.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} \end{bmatrix}, \text{ and}$$
$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} \\ \mathbf{B}_{1,0} & \mathbf{B}_{1,1} \end{bmatrix}.$$

Each $\mathbf{A}_{i,j}$ and $\mathbf{B}_{i,j}$ is a matrix of size 1000×1000, with entries chosen from the following distributions. $\mathbf{A}_{0,0}$, $\mathbf{A}_{0,1}$ are distributed Unif$(0, \ldots, 9999)$ and $\mathbf{A}_{1,0}$, $\mathbf{A}_{1,1}$ distributed Unif$(0, \ldots, 9)$. Next, $\mathbf{B}_{0,0}$, $\mathbf{B}_{0,1}$ are distributed Unif$(0, \ldots, 9)$ and $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ distributed Unif$(0, \ldots, 9999)$. In this scenario, the DRC requires us to multiply each matrix by 0.1 and quantize each entry between 0 and 999. Note that this implies that $\mathbf{A}_{1,0}, \mathbf{A}_{1,1}, \mathbf{B}_{0,0}, \mathbf{B}_{0,1}$ are all quantized into zero submatrices since the entry in these four submatrices is less than 10. We label the quantized matrices by the superscript $\tilde{\cdot}$. We emphasize that the finite field embedding technique *only* recovers the product of these quantized matrices. However, this product is

$$\tilde{\mathbf{A}}^T\tilde{\mathbf{B}} = \begin{bmatrix} \tilde{\mathbf{A}}_{0,0} & \tilde{\mathbf{A}}_{0,1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \tilde{\mathbf{B}}_{1,0} & \tilde{\mathbf{B}}_{1,1} \end{bmatrix} = \mathbf{0}.$$

Thus, the final estimate of the original product $\mathbf{A}^T\mathbf{B}$, denoted as $\widehat{\mathbf{A}^T\mathbf{B}}$ is the all-zeros matrix. This implies that the normalized MSE of their scheme is exactly 1.0. Thus, the performance of the finite field embedding technique has a strong dependence on the matrix entries. We note here that even if we consider other quantization schemes or larger 64-bit primes, one can arrive at adversarial examples such as the ones shown above. Once again for these examples, our methods have a normalized MSE of at most 10^{-27}.

In our experience, the finite field embedding technique also suffers from significant computational issues in implementation. Note that the technique requires the computation of the inverse matrix at the master node that is required for decoding

TABLE II

PERFORMANCE OF MATRIX INVERSION OVER A LARGE PRIME ORDER FIELD IN PYTHON 3.7. THE TABLE SHOWS THE COMPUTATION TIME FOR INVERTING A $\ell \times \ell$ MATRIX $\mathbf{G}$ OVER A FINITE FIELD OF ORDER p. LET $\widehat{\mathbf{G}^{-1}}$ DENOTE THE INVERSE OBTAINED BY APPLYING THE SYMPY FUNCTION `Matrix(G) .inverse_mod(p)`. THE MSE IS DEFINED AS $\frac{1}{\ell}||\mathbf{G}\widehat{\mathbf{G}^{-1}} - \mathbf{I}||_F$

ℓ	p	Computation Time (s)	MSE
9	65537	1.39	0
12	65537	4.38	0
15	65537	12.64	0
9	2147483647	1.39	0
12	2147483647	4.68	1.8×10^9
15	2147483647	14.45	4.2×10^9

the final result. We implemented this within the Python 3.7, sympy library (see [21] Git hub repository). We performed experiments with $p = 65537$ and $p = 2147483647$. As shown in Table II, for the smaller prime $p = 65537$, the inverse computation is accurate up to 15×15 matrices; however, the computation time of the inverse is rather high and can dominate the overall execution time. On the other hand for the larger prime $p = 2147483647$, the error in in the computed inverse is very high for 12×12 and 15×15 matrices; the corresponding time taken is even higher. It is possible that very careful implementations can perhaps avoid these issues. However, we are unaware of any such publicly available code. To summarize, the finite field embedding technique suffers from major dynamic range limitations and associated computational issues and cannot be used to support real computations.

The work most closely related to ours is by [15], which demonstrates an upper bound of $O(q^{2(q-\tau)})$ on the worst-case condition number. It can be noted that this grows much faster than our upper bound in the parameter $q - \tau$. In numerical experiments, our worst-case condition numbers are much smaller than the work of [15]; we discuss this in the upcoming Section VI-A. We note that the results in [15] are given in terms of the condition number calculated using the Frobenius norm,[1] i.e., for matrix $\mathbf{M}$, they define $\kappa(\mathbf{M}) = ||\mathbf{M}||_F||\mathbf{M}^{-1}||_F$. However, there are well-known relations between different matrix norms. In particular when $\mathbf{M}$ is of size $\ell \times \ell$, then $||\mathbf{M}||_2 \leq ||\mathbf{M}||_F \leq \sqrt{\ell}||\mathbf{M}||_2$. This allows us to compare the corresponding Frobenius-norm induced condition number as well.

Both our scheme and [15] have the optimal threshold when $\mathbf{A}$ and $\mathbf{B}$ are only divided into block-columns (*cf.* Section IV)). However, when the matrices are split across both rows and columns (*cf.* Section V) the polynomial code approach of [3] has a lower threshold of $pk_Ak_B + p - 1$, while our threshold is $2pk_Ak_B - 1$; the thresholds match when $k_A = k_B = 1$. The work of [15] in this scenario, i.e., when $p > 1$ has a threshold denoted τ_{F-C} given by

$$\tau_{F-C} = 4k_Ak_Bp - 2(k_Ak_B + pk_A + pk_B) + k_A + k_B + 2p - 1.$$

It can be seen that if $k_A = 1$ or $k_B = 1$, then $\tau_{F-C} \leq 2pk_Ak_B - 1$. However, when $k_A > 1$ and $k_B > 1$, simple analysis shows that our threshold $\leq \tau_{F-C}$ (see Claim 4 in the Appendix).

Certain approaches [11]–[13], [22] only apply for matrix-vector multiplication and furthermore do not provide any explicit guarantees on the worst-case condition number. Other approaches include the work of [16] which uses random linear encoding of the $\mathbf{A}$ and $\mathbf{B}$ matrices and the work of [14] that uses a convolutional coding approach to this problem. Both these approaches require random sampling and do not have a theoretical upper bound on the worst-case condition number. However, for a given set of random choices, it is possible to numerically compute an upper bound on the worst-case condition number of [14].

[1]For measuring the error in decoding a system of equations corresponding to $\mathbf{M}$ it is more natural to consider an induced norm, like the one we use.

A. Numerical Experiments

The central point of our work is that we can leverage the well-conditioned behavior of Vandermonde matrices with parameters on the unit circle while continuing to work with computation over the reals. We compare our results with the work of [6] (called "Real Vandermonde"), a "Complex Vandermonde" scheme where the evaluation points are chosen from the complex unit circle, the work of [14], [15] and [16]. For the normalized MSE simulations below, we always pick the set of worker nodes that correspond to the worst-case condition number of the corresponding method. Additive Gaussian noise is added to the encoded matrix and vector in the matrix-vector case and both encoded matrices in the matrix-matrix case (details in [23]).

All experiments were run on the AWS EC2 system with a t2.2xlarge instance (for master node) and t2.micro instances (for slave nodes). The source code can be found in [23].

1) Matrix-Vector Case: In Table III, we compare the *average* and *worst-case* condition number of the different schemes for matrix-vector multiplication. The system under consideration has $n = 31$ worker nodes and a threshold specified by the third column (labeled as τ). The evaluation points for [6] were uniformly sampled from the interval $[-1, 1]$ [24]. The Complex Vandermonde scheme has evaluation points which are the 31-st root of unity. The [15] and [16] schemes are not applicable for the matrix-vector case. It can be observed from Table III that both the worst-case and the average condition numbers of our scheme are over eleven orders of magnitude better than the Real Vandermonde scheme. Furthermore, there is an exact match of the condition number values for all the other schemes. This can be understood by following the discussion in Section IV-B. Specifically, our schemes have the property that the condition number only depends on the eigenvalues of corresponding circulation permutation matrix and rotation matrix respectively. These eigenvalues lie precisely within 31-th roots of unity. The methods of [14] have some divisibility constraints on the number of columns in $\mathbf{A}$. Accordingly, we considered a matrix with 21924 columns for it. We performed 200 random trials for picking the best Random Conv. code [14]. The worst-case condition number of these methods are still around one to two orders of magnitude higher than ours.

It can be observed that the decoding flop count for both matrix-vector and matrix-matrix multiplication is independent of t, i.e., in the regime where t is very large the decoding time may be neglected with respect to the worker node computation time. Nevertheless, from a practical perspective it is useful to understand the decoding times as well.

When the matrix $\mathbf{A}$ is of dimension 28000×19720 and $\mathbf{x}$ is of length 28000, the last two columns in Table III indicate the average worker node computation time and the master node decoding time for the different schemes. These numbers were obtained by averaging over several runs of the algorithm. It can be observed that the Complex Vandermonde scheme requires about twice the worker computation time as our schemes. Thus, it is wasteful of worker node computation resources. On the other hand, our schemes leverage the same condition number with computation over the reals. The decoding times

TABLE III

COMPARISON FOR MATRIX-VECTOR CASE WITH $n = 31$, $\mathbf{A}$ HAS SIZE 28000×19720 AND $\mathbf{x}$ HAS LENGTH 28000 FOR THE FIRST FOUR METHODS. FOR THE ALL ONES CONV. AND RANDOM CONV. (FROM [14]), $\mathbf{A}$ HAS 21924 COLUMNS

Scheme	γ_A	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Real Vand.	$1/29$	29	1.1×10^{13}	2.9×10^{13}	1.2×10^{-3}	9×10^{-5}
Complex Vand.	$1/29$	29	12	55	2.9×10^{-3}	2.8×10^{-4}
Circ. Perm. Embed.	$1/28$	29	12	55	1.2×10^{-3}	3.7×10^{-4}
Rot. Mat. Embed.	$1/29$	29	12	55	1.3×10^{-3}	10^{-4}
All Ones Conv. [14]	$1/27$	29	1386	5093	1.4×10^{-3}	9×10^{-4}
Random Conv. [14]	$1/27$	29	259	4903	1.4×10^{-3}	5×10^{-4}

TABLE IV

COMPARISON FOR $\mathbf{A}^T\mathbf{B}$ MATRIX-MATRIX MULTIPLICATION CASE WITH $n = 31$, $k_A = 4$, $k_B = 7$. $\mathbf{A}$ HAS SIZE 8000×14000, $\mathbf{B}$ HAS SIZE 8400×14000

Scheme	γ_A	γ_B	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Real Vand.	$1/4$	$1/7$	28	4.9×10^{12}	2.3×10^{13}	2.132	0.407
Complex Vand.	$1/4$	$1/7$	28	27	404	8.421	1.321
Rot. Mat. Embed.	$1/4$	$1/7$	28	27	404	2.121	0.408
Ortho-Poly [15]	$1/4$	$1/7$	28	1449	8.3×10^4	2.263	0.412
RKRP [16]	$1/4$	$1/7$	28	255	5.6×10^4	2.198	0.406
Random Conv. [14]	$1/3$	$1/6$	28	-	$\leq 3.4 \times 10^4$	-	-

of almost all the schemes are quite small. However, the Circulant Permutation Matrix scheme requires decoding time which is somewhat higher than the rotation matrix embedding even though we can use FFT based approaches for it. We expect that for much larger scale problems, the FFT based approach may be faster.

Our next set of results compare the mean-squared error (MSE) in the decoded result for the different schemes. To simulate numerical precision problems, we added i.i.d Gaussian noise (of different SNRs) to the encoded submatrices of $\mathbf{A}$ and the vector $\mathbf{x}$ (the encoded submatrices of $\mathbf{B}$) in each worker node. The master node then performs decoding on the noisy vectors. The plots in Figure 1 correspond to the worst-case choice of worker nodes for each of the schemes. It can be observed that the Circulant Permutation Matrix Embedding has the best performance. This is because many of the matrices on the block-diagonal in (13) (see Section B in the appendix) have well-behaved condition numbers and only a few correspond to the worst-case. We have not shown the results for the Real Vandermonde case here because the normalized MSE was very large.

2) Matrix-Matrix Case: In the matrix-matrix scenario we again consider a system with $n = 31$ worker nodes and $k_A = 4$ and $k_B = 7$ so that the threshold $\tau = k_A k_B = 28$. Once again we observe (*cf.* Table IV) that the worst-case condition number of the Rotation Matrix Embedding is about eleven orders of magnitude lower than the Real Vandermonde case. Furthermore, the schemes of [15] and [16] have a worst-case condition numbers that are two orders of magnitude higher than our scheme. For both [16] and [14] schemes we performed 200 random trials and picked the scheme with the lowest worst-case condition number. For [14], we only report the upper bound on the worst-case condition number. Finding the actual worst-case recovery set takes a long time.

When the matrix $\mathbf{A}$ is of dimension 8000×14000 and $\mathbf{B}$ is of dimension 8000×14000, the worker node computation

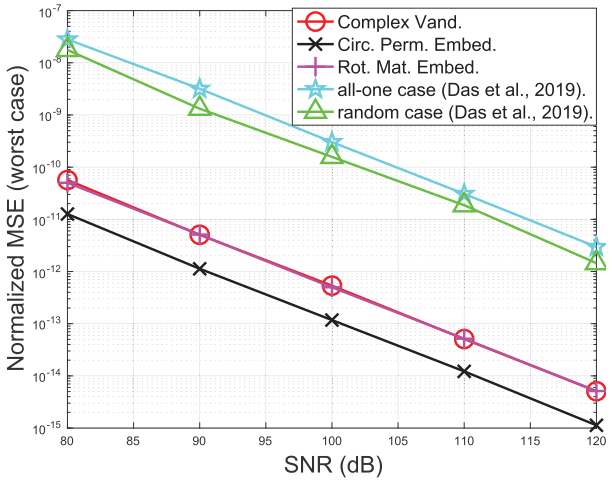

Fig. 1. Consider matrix-vector $\mathbf{A}^T\mathbf{x}$ multiplication system with $n = 31$, $\tau = 29$. $\mathbf{A}$ has size 28000×19720 and $\mathbf{x}$ has length 28000.

times and decoding times are listed in Table IV. As expected the Complex Vandermonde scheme takes much longer for the worker node computations, whereas the Rotation Matrix Embedding, [15] and [16] take about the same time. The decoding times are also very similar. As shown in Figure 2, the normalized MSE of our Rotation Matrix Embedding scheme is much about five orders of magnitude lower than the scheme of [15]. The normalized MSE of the Real Vandermonde case is very large so we do not plot it. Since we did not determine the worst-case recovery set for [14], we have not included the data and corresponding curves for it.

In the matrix-matrix multiplication scenario with $p \geq 2$, we consider a system with $n = 17$ worker nodes and $u_A = 2, u_B = 2, p = 2$. Note that in this case the threshold of [3] is lower than our threshold and [15]. Accordingly, we picked a setting where the our and [15]'s threshold match and only compare these results.

TABLE V

COMPARISON FOR MATRIX-MATRIX $\mathbf{A}^T\mathbf{B}$ MULTIPLICATION CASE WITH $n = 17$, $u_A = 2$, $u_B = 2$, $p = 2$, $\mathbf{A}$ IS OF SIZE 4000×16000, $\mathbf{B}$ IS OF 4000×16000

Scheme	γ_A	γ_B	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Rot. Mat. Embed.	$1/4$	$1/4$	15	7	22	2.23	0.69
Ortho-Poly [15]	$1/4$	$1/4$	15	10^4	2.7×10^5	2.23	0.18

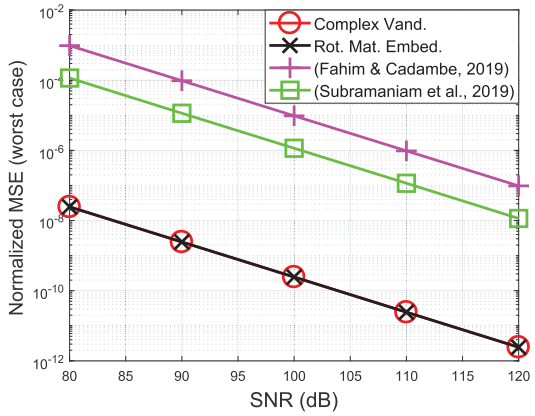

Fig. 2. Consider matrix-matrix $\mathbf{A}^T\mathbf{B}$ multiplication system with $n = 31$, $k_A = 4$, $k_B = 7$, $\mathbf{A}$ is of size 8000×14000, $\mathbf{B}$ is of 8400×14000.

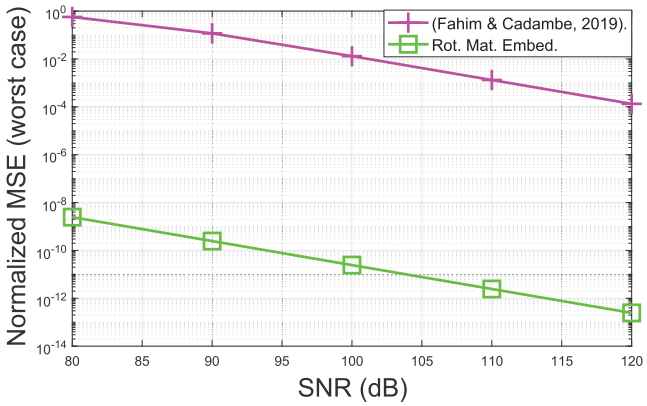

Fig. 3. Consider matrix-matrix $\mathbf{A}^T\mathbf{B}$ multiplication system with $n = 18$, $u_A = 2$, $u_B = 2$, $p = 2$, $\mathbf{A}$ is of size 4000×16000, $\mathbf{B}$ is of 4000×16000.

We observe that the condition number of the Rotation Matrix Embedding scheme is about four orders of magnitude lower than [15]. Figure 3 shows that the normalized MSE of our Rotation Matrix Embedding scheme is much lower than [15]. The Rotation Matrix Embedding scheme has higher decoding time since its decoding algorithm operates over the complex field.

VII. CONCLUSION AND FUTURE WORK

In this work we demonstrated that polynomial based schemes for coded computation suffer from serious numerical stability issues in practice. This stems from the provably bad conditioning of real Vandermonde matrices. We demonstrated a technique that exploits the properties of circulant and rotation matrices for coded computation. In essence, our method allows us to leverage the superior conditioning of complex Vandermonde matrices with parameters on the unit circle while still working with real computations at the worker nodes. The worst-case condition number of our recovery matrices is upper bounded by $O(n^{s+5.5})$ (where n- number of workers, s- number of stragglers) and our schemes have excellent performance in numerical experiments.

It is to be noted that our upper bound grows with the number of stragglers. In fact, it can be shown that if s is a large fraction of n, then the condition number of the corresponding recovery matrices can be quite large even in the complex Vandermonde on unit circle case. It would be interesting to investigate coded computation schemes that continue to be numerically stable in the large s regime.

APPENDIX

A. Proof of Claim 1

Proof: Note that Algorithm 2 is applied for recovering the corresponding entries of $\mathbf{A}_{i,j}^T\mathbf{x}$ for $i \in [k_A], j \in [\tilde{q}]$ separately. There are $r/(k_A(q-1))$ such entries. The complexity of computing a N-point FFT is $O(N \log N)$ in terms of the required floating point operations (flops). Computing the permutation does not cost any flops and its complexity is negligible as compared to the other steps. Step 1 of Algorithm 2 therefore has complexity $O(k_A\tilde{q}\log\tilde{q})$. In Step 2, we solve the degree $k_A - 1$ polynomial interpolation, $(\tilde{q} - 1)$ times. This takes $O((\tilde{q}-1)k_A \log^2 k_A)$ time [25]. Finally, Step 3, requires applying the inverse permutation and the inverse FFT; this requires $O(k_A\tilde{q}\log\tilde{q})$ operations. Therefore, the overall complexity is given by

$$\frac{r}{k_A(\tilde{q}-1)}\left(O(k_A\tilde{q}\log\tilde{q}) + O((\tilde{q}-1)k_A\log k_A^2)\right)$$
$$\approx O(r(\log\tilde{q} + \log^2 k_A)).$$

□

B. Proof of Theorem 3

Proof: The arguments are conceptually similar to the proof of Theorem 2. Suppose that the workers indexed by $i_0, \ldots, i_{k_A-1}$ complete their tasks. The corresponding block-columns of $\mathbf{G}^{circ}$ can be extracted to form

$$\tilde{\mathbf{G}} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{P}^{i_0} & \mathbf{P}^{i_1} & \cdots & \mathbf{P}^{i_{k_A-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{i_0(k_A-1)} & \mathbf{P}^{i_1(k_A-1)} & \cdots & \mathbf{P}^{i_{k_A-1}(k_A-1)} \end{bmatrix}.$$

As in the proof of Theorem 2 we can equivalently analyze the decoding by considering the system of equations

$$\mathbf{m}\tilde{\mathbf{G}} = \mathbf{c},$$

where $\mathbf{m}, \mathbf{c} \in \mathbb{R}^{1\times k_A\tilde{q}}$ are row-vectors such that

$$\begin{aligned}\mathbf{m} =&[\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}]\\ =&[\mathbf{m}_{\langle 0,0\rangle}, \cdots, \mathbf{m}_{\langle 0,\tilde{q}-1\rangle}, \cdots,\\ &\cdots \mathbf{m}_{\langle k_A-1,0\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,\tilde{q}-1\rangle}], \text{ and}\\ \mathbf{c} =&[\mathbf{c}_{i_0}, \cdots, \mathbf{c}_{i_{k_A-1}}]\\ =&[\mathbf{c}_{\langle i_0,0\rangle}, \cdots, \mathbf{c}_{\langle i_0,\tilde{q}-1\rangle}, \cdots,\\ &\cdots, \mathbf{c}_{\langle i_{k_A-1},0\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},\tilde{q}-1\rangle}].\end{aligned}$$

Note that not all variables in $\mathbf{m}$ are independent owing to (7). Let $\mathbf{m}^{\mathcal{F}}$ and $\mathbf{c}^{\mathcal{F}}$ denote the $\tilde{q}$-point "block-Fourier" transforms of these vectors, i.e,

$$\mathbf{m}^{\mathcal{F}} = \mathbf{m}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix} \text{ and}$$

$$\mathbf{c}^{\mathcal{F}} = \mathbf{c}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix},$$

where $\mathbf{W}$ is the $\tilde{q}$-point DFT matrix. Let $\tilde{\mathbf{G}}_{k,l} = \mathbf{P}^{i_l k}$ denote the (k,l)-th block of $\tilde{\mathbf{G}}$. Using the fact that $\mathbf{P}$ can be diagonalized by the DFT matrix $\mathbf{W}$, we have

$$\tilde{\mathbf{G}}_{k,l} = \mathbf{W}\text{diag}(1, \omega_{\tilde{q}}^{i_l k}, \omega_{\tilde{q}}^{2i_l k}, \ldots, \omega_{\tilde{q}}^{(\tilde{q}-1)i_l k})\mathbf{W}^*.$$

Let $\tilde{\mathbf{G}}^{\mathcal{F}}_{k,l} = \text{diag}(1, \omega_{\tilde{q}}^{i_l k}, \omega_{\tilde{q}}^{2i_l k}, \ldots, \omega_{\tilde{q}}^{(\tilde{q}-1)i_l k})$, and $\tilde{\mathbf{G}}^{\mathcal{F}}$ represent the $k_A \times k_A$ block matrix with $\tilde{\mathbf{G}}^{\mathcal{F}}_{k,l}$ for $k, l = 0, \ldots, k_A - 1$ as its blocks. Therefore, the system of equations

$$\mathbf{m}\tilde{\mathbf{G}} = \mathbf{c},$$

can be further expressed as

$$\begin{aligned}&\mathbf{m}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{W}^* & & \\ & \ddots & \\ & & \mathbf{W}^*\end{bmatrix}\tilde{\mathbf{G}}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix}\\ &= \mathbf{c}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix},\\ &\implies [\mathbf{m}_0^{\mathcal{F}}, \cdots, \mathbf{m}_{k_A-1}^{\mathcal{F}}]\tilde{\mathbf{G}}^{\mathcal{F}} = [\mathbf{c}_{i_0}^{\mathcal{F}}, \cdots, \mathbf{c}_{i_{k_A-1}}^{\mathcal{F}}]\end{aligned}$$

upon right multiplication by the matrix $\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix}$. Next, we note that as each block within $\tilde{\mathbf{G}}^{\mathcal{F}}$ has a diagonal structure, we can rewrite the system of equations as a block diagonal matrix upon applying an appropriate permutation (*cf.* Claim 2 in Section E in the appendix). Thus, we can rewrite it as

$$[\mathbf{m}_0^{\mathcal{F},\pi}, \cdots, \mathbf{m}_{\tilde{q}-1}^{\mathcal{F},\pi}]\tilde{\mathbf{G}}_d^{\mathcal{F}} = [\mathbf{c}_0^{\mathcal{F},\pi}, \cdots, \mathbf{c}_{\tilde{q}-1}^{\mathcal{F},\pi}], \tag{12}$$

where the permutation π is such that $\mathbf{m}_j^{\mathcal{F},\pi} = [\mathbf{m}_{0,j}^{\mathcal{F}} \ \mathbf{m}_{1,j}^{\mathcal{F}} \ \ldots \ \mathbf{m}_{k_A-1,j}^{\mathcal{F}}]$ and likewise $\mathbf{c}_j^{\mathcal{F},\pi} = [\mathbf{c}_{i_0,j}^{\mathcal{F}} \ \mathbf{c}_{i_1,j}^{\mathcal{F}} \ \ldots \ \mathbf{c}_{i_{k_A-1},j}^{\mathcal{F}}]$. Furthermore, $\tilde{\mathbf{G}}_d^{\mathcal{F}}$ is a block-diagonal matrix where each block is of size $k_A \times k_A$. Now, according to (7), we have $\mathbf{m}_{i,0}^{\mathcal{F}} = \sum_{j=0}^{\tilde{q}-1}\mathbf{m}_{i,j} = 0$ for $i = 0, \ldots, k_A - 1$, which implies that $\mathbf{m}_0^{\mathcal{F},\pi}$ is a $1 \times k_A$ zero row-vector and thus $\mathbf{c}_0^{\mathcal{F},\pi}$ is too.

In what follows, we show that each of the other diagonal blocks of $\tilde{\mathbf{G}}_d^{\mathcal{F}}$ is non-singular. This means that $[\mathbf{m}_0^{\mathcal{F}}, \cdots, \mathbf{m}_{k_A-1}^{\mathcal{F}}]$ and consequently $\mathbf{m}$ can be determined by solving the system of equations in (12). Towards this end, we note that the k-th diagonal block $(1 \le k \le \tilde{q}-1)$ of $\tilde{\mathbf{G}}_d^{\mathcal{F}}$, denoted by $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ can be expressed as follows.

$$\tilde{\mathbf{G}}_d^{\mathcal{F}}[k] = \begin{bmatrix}1 & 1 & \cdots & 1\\ \omega_{\tilde{q}}^{i_0 k} & \omega_{\tilde{q}}^{i_1 k} & \cdots & \omega_{\tilde{q}}^{i_{k_A-1} k}\\ \vdots & \vdots & \ddots & \vdots\\ \omega_{\tilde{q}}^{(k_A-1)i_0 k} & \omega_{\tilde{q}}^{(k_A-1)i_1 k} & \cdots & \omega_{\tilde{q}}^{(k_A-1)i_{k_A-1} k}\end{bmatrix}. \tag{13}$$

The above matrix is a complex Vandermonde matrix with parameters $\omega_{\tilde{q}}^{i_0 k}, \ldots, \omega_{\tilde{q}}^{i_{k_A-1} k}$. Thus, as long these parameters are distinct, $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ will be non-singular. Note that we need the property to hold for $k = 1, \ldots, \tilde{q}-1$. This condition can be expressed as

$$(i_\alpha - i_\beta)k \not\equiv 0 \pmod{\tilde{q}},$$

for $i_\alpha, i_\beta \in \{0, \ldots, n-1\}$ and $1 \le k \le \tilde{q}-1$. A necessary and sufficient condition for this to hold is that $\tilde{q}$ is prime. An application of Theorem 1 shows that $\kappa(\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]) \le O(\tilde{q}^{\tilde{q}-k_A+c_1})$ for all k. As decoding $\mathbf{m}$ is equivalent to solving systems of equations specified by $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ for $1 \le k \le \tilde{q}-1$, the worst-case condition number is at most $O(\tilde{q}^{\tilde{q}-k_A+c_1})$. □

C. Vandermonde Matrix Condition Number Analysis

Let $\mathbf{V}$ be a $m \times m$ Vandermonde matrix with parameters $s_0, s_1, \ldots s_{m-1}$. We are interested in upper bounding $\kappa(\mathbf{V})$. Let $s_+ = \max_{i=0}^{m-1}|s_i|$. Then, it is known that $||\mathbf{V}|| \le m\max(1, s_+^{m-1})$ [9]. Finding an upper bound on $||\mathbf{V}^{-1}||$ is more complicated and we discuss this in detail below. Towards this end we need the definition of a Cauchy matrix.

Definition 5: A $m \times m$ Cauchy matrix is specified by parameters $\mathbf{s} = [s_0 \ s_1 \ \ldots \ s_{m-1}]$ and $\mathbf{t} = [t_0 \ t_1 \ \ldots \ t_{m-1}]$, such that its (i,j)-th entry

$$\mathbf{C}_{\mathbf{s},\mathbf{t}}(i,j) = \left(\frac{1}{s_i - t_j}\right) \text{ for } i \in [m], j \in [m].$$

In what follows, we establish an upper bound on the condition number of Vandermonde matrices with parameters on the unit circle.

Proof of Theorem 1:

Proof: Recall that $\omega_q = e^{\mathrm{i}\frac{2\pi}{\mathrm{q}}}$ and $\omega_m = e^{\mathrm{i}\frac{2\pi}{\mathrm{m}}}$ and define $t_j = f\omega_m^j, j = 0, \ldots, m-1$ where f is a complex number with $|f| = 1$. We let $\mathbf{C}_{\mathbf{s},f}$ denote the Cauchy matrix with parameters $\{s_0, \ldots, s_{m-1}\}$ and $\{t_0, \ldots, t_{m-1}\}$. Let $\mathbf{W}$ be the m-point DFT matrix. The work of [9] shows that

$$\begin{aligned}\mathbf{V}^{-1} =&\text{diag}(f^{m-1-j})_{j=0}^{m-1}\sqrt{m}\mathbf{W}^*\\ &\text{diag}(\omega_m^{-j})_{j=0}^{m-1}\mathbf{C}_{\mathbf{s},f}^{-1}\text{diag}\left(\frac{1}{s_j^m - f^m}\right)_{j=0}^{m-1}.\end{aligned}$$

It can be seen that the matrix $\text{diag}(f^{m-1-j})_{j=0}^{m-1}\mathbf{W}^*\text{diag}(\omega_m^{-j})_{j=0}^{m-1}$ is unitary. Therefore,

$$\begin{aligned}
&||\mathbf{V}^{-1}|| \\
=&\sqrt{m}||\mathbf{C}_{\mathbf{s},f}^{-1}\text{diag}\left(\frac{1}{s_j^m - f^m}\right)_{j=0}^{m-1}|| \\
\leq&\sqrt{m}||\mathbf{C}_{\mathbf{s},f}^{-1}|| \times \left(\frac{1}{\min_{i=0}^{m-1}|s_i^m - f^m|}\right) \\
\leq&m^{1.5} \times (\max_{i',j'}|\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')|) \times \left(\frac{1}{\min_{i=0}^{m-1}|s_i^m - f^m|}\right),
\end{aligned} \tag{14}$$

where the first inequality holds as the norm of a product of matrices is upper bounded by the products of the individual norms and second inequality holds since for any $\mathbf{M}$, we have $||\mathbf{M}|| \leq ||\mathbf{M}||_F$.

In what follows, we upper bound the RHS of (14). Let $s(x)$ denote a function of x so that $s(x) = \Pi_{i=0}^{m-1}(x - s_i)$. The (i',j')-the entry of $\mathbf{C}_{\mathbf{s},f}^{-1}$ can be expressed as [9]

$$\begin{aligned}
&\mathbf{C}_{\mathbf{s},f}^{-1}(i',j') = (-1)^m s(t_{j'})(s_{i'}^m - f^m)/(s_{i'} - t_{j'}), \text{ so that} \\
&|\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')| = |s(t_{j'})||s_{i'}^m - f^m|/|s_{i'} - t_{j'}| \\
&\qquad\leq |s(t_{j'})|(|s_{i'}^m| + |f^m|)/|s_{i'} - t_{j'}| \\
&\qquad= 2|s(t_{j'})|/|s_{i'} - t_{j'}| \quad (\text{since } |s_{i'}| = |f| = 1).
\end{aligned}$$

Let $\mathcal{M} = \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\} \setminus \{s_0, s_1, \ldots, s_{m-1}\}$ denote the q-th roots of unity that are *not* parameters of $\mathbf{V}$. Note that

$$\begin{aligned}
s(t_{j'}) &= \Pi_{i=0}^{m-1}(t_{j'} - s_i) \\
&= \left.\frac{x^q - 1}{\Pi_{\alpha_j \in \mathcal{M}}(x - \alpha_j)}\right|_{x=t_{j'}}, \text{ so that} \\
|s(t_{j'})| &= \frac{|t_{j'}^q - 1|}{\Pi_{\alpha_j \in \mathcal{M}}|t_{j'} - \alpha_j|} \\
&\leq \frac{2}{\Pi_{\alpha_j \in \mathcal{M}}|t_{j'} - \alpha_j|} \\
&\quad(\text{since } |t_{j'}| = 1 \text{ and by the triangle inequality}).
\end{aligned}$$

Thus, we can conclude that

$$\begin{aligned}
&\max_{i',j'}|\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')| \\
\leq&4\max_{i',j'}\frac{1}{\Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)|}\frac{1}{|s_{i'} - t_{j'}|} \\
=&4\left(\frac{1}{\min_{i',j'}\Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)|}\frac{1}{|s_{i'} - t_{j'}|}\right).
\end{aligned} \tag{15}$$

Note that in the expression above the α_j's and $s_{i'}$ are all points within $\Omega_q = \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$. We choose $f = e^{\mathrm{i}\frac{\pi}{m}}$ so that $t_{j'} = f\omega_m^{j'} = e^{\mathrm{i}\frac{\pi}{m}}\omega_m^{j'}$. Now for any i' and j' we need to lower bound $\Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)||s_{i'} - t_{j'}|$. Towards this end, we note that the distance between two points on the unit circle can be expressed as $2\sin(\theta/2)$ if θ is the induced angle between them. Furthermore, we have $2\sin(\theta/2) \geq 2\theta/\pi$ as long as $\theta \leq \pi$.

Let $d = q - m$. Then, for any choice of $t_{j'}$ we can consider lower bounds on the distances of $d+1$ points that lie on Ω_q. It can be seen that the closest point to $t_{j'}$ that lies within Ω_q has an induced angle

$$\left|\frac{2\pi\ell}{q} - \frac{2\pi(j' + \frac{1}{2})}{m}\right| \geq \frac{2\pi}{qm}\frac{1}{2} \geq \frac{\pi}{q^2} \text{ (since } q \text{ is odd \& } q > m).$$

Therefore, the corresponding distance is lower bounded by $2/q^2$. Similarly, the next closest distance is lower bounded by $2/q$, followed by $2(2/q), 3(2/q), \ldots, d(2/q)$. Then,

$$\begin{aligned}
&\min_{i',j'}\left(\Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)|\right)|s_{i'} - t_{j'}| \\
&\geq 2/q^2 \times 2/q \times 4/q \times \cdots \times 2d/q \\
&= 2^{d+1}d!\frac{1}{q^{d+2}}.
\end{aligned}$$

Therefore,

$$\max_{i',j'}|\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')| \leq \frac{q^{d+2}}{C_d}$$

where $C_d = 2^{d-1}d!$ is a constant. Let the i-th parameter $s_i = e^{\mathrm{i}2\pi\ell/\mathrm{q}}$. Then,

$$\begin{aligned}
|s_i^m - f^m| &= |e^{\mathrm{i}2\pi\ell\mathrm{m}/\mathrm{q}} + 1| \\
&= 2|\cos(\pi\ell m/q)|.
\end{aligned}$$

The term ℓm can be expressed as $\ell m = \beta q + \eta$ for integers β and η such that $0 \leq \eta \leq q - 1$. Now note that $\eta \neq q/2$ since by assumption q is odd. Thus, $|\cos(\pi\ell m/q)|$ takes its smallest value when $\eta = (q+1)/2$ or $(q-1)/2$. In this case

$$\begin{aligned}
|\cos(\pi\ell m/q)| &= \left|\cos\left(\beta\pi + \pi\frac{q+1}{2q}\right)\right| \\
&\geq \left|\sin\left(\frac{\pi}{2q}\right)\right| \\
&\geq \frac{1}{q}.
\end{aligned}$$

Thus, we can upper bound the RHS of (14) and obtain

$$\begin{aligned}
||\mathbf{V}^{-1}|| &\leq m^{1.5}\frac{q^{d+2}}{C_d}q \\
&\leq \frac{q^{d+4.5}}{C_d} \text{ (since } m < q).
\end{aligned}$$

Finally, using the fact that $||V|| \leq m < q$. we obtain

$$\kappa(\mathbf{V}) \leq \frac{q^{d+5.5}}{C_d}.$$

□

D. Proof of Theorem 5

Proof: We proceed in a similar manner as in Example 4. Following the encoding rules (*cf.* Algorithm 4) and worker computation rules (*cf.* (10)), we can analyze the computation in worker k as follows. Let $(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{(\langle i,0\rangle,j)} \\ \mathbf{A}_{(\langle i,1\rangle,j)}\end{bmatrix} = \begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \mathbf{A}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\end{bmatrix}$ and $(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{(\langle i,0\rangle,j)} \\ \mathbf{B}_{(\langle i,1\rangle,j)}\end{bmatrix} = \begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\end{bmatrix}$. Let $\hat{\mathbf{A}}_k = \begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle} \\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}$

and $\hat{\mathbf{B}}_k = \begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}$. Then, we have

$$\begin{aligned}
\hat{\mathbf{A}}_k^{\mathcal{F}} &= (\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\hat{\mathbf{A}}_k \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} (\mathbf{Q}^*\mathbf{R}_{-\theta}^{k((j-1)p+i+1)}\mathbf{Q}\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{(\langle i,0\rangle,j)}\\ \mathbf{A}_{(\langle i,1\rangle,j)}\end{bmatrix} \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} (\Lambda^{*k((j-1)p+i+1)} \otimes \mathbf{I}_\zeta)(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{(\langle i,0\rangle,j)}\\ \mathbf{A}_{(\langle i,1\rangle,j)}\end{bmatrix} \\
&= \begin{bmatrix}\sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}\omega_q^{*k((j-1)p+i+1)}\mathbf{A}^{\mathcal{F}}_{(\langle i,0\rangle,j)}\\ \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}\omega_q^{*-k((j-1)p+i+1)}\mathbf{A}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\end{bmatrix}, \text{ and} \\
\hat{\mathbf{B}}_k^{\mathcal{F}} &= (\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\hat{\mathbf{B}}_k \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} (\mathbf{Q}^*\mathbf{R}_{\theta}^{k(p-1-i+jpk_A)}\mathbf{Q}\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{(\langle i,0\rangle,j)}\\ \mathbf{B}_{(\langle i,1\rangle,j)}\end{bmatrix} \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} (\Lambda^{k(p-1-i+jpk_A)} \otimes \mathbf{I}_\zeta)(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)}\\ \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\end{bmatrix} \\
&= \begin{bmatrix}\sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1}\omega_q^{k(p-1-i+jpk_A)}\mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)}\\ \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}\omega_q^{-k(p-1-i+jpk_A)}\mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\end{bmatrix}.
\end{aligned}$$

This implies that

$$\begin{aligned}
\hat{\mathbf{A}}_k^T\hat{\mathbf{B}}_k =& ((\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\hat{\mathbf{A}}_k)^*(\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\hat{\mathbf{B}}_k \\
=& \hat{\mathbf{A}}_k^{\mathcal{F}*}\hat{\mathbf{B}}_k^{\mathcal{F}} \\
=& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}\omega_q^{k((j-1)p+i+1)}\mathbf{A}^{\mathcal{F}*}_{(\langle i,0\rangle,j)}\right) \\
& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1}\omega_q^{k(p-1-i+jpk_A)}\mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)}\right) + \\
& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1}\omega_q^{-k((j-1)p+i+1)}\mathbf{A}^{\mathcal{F}*}_{(\langle i,1\rangle,j)}\right) \\
& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1}\omega_q^{-k(p-1-i+jpk_A)}\mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\right). \qquad (16)
\end{aligned}$$

To better understand the behavior of the sum in (16), we divide it into the following two cases.

- *Case 1: Useful terms.* The master node wants to recover $\mathbf{C} = \mathbf{A}^T\mathbf{B} = [\mathbf{C}_{i,j}]$, $i \in [k_A]$, $j \in [k_B]$, where each $\mathbf{C}_{i,j}$ is a block matrix of size $r/k_A \times w/k_B$. Note that $\mathbf{C}_{i,j} = \sum_{u=0}^{p-1}(\mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)} + \mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)})$. Moreover, note that

$$\begin{aligned}
&\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)} + \mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)} \\
=& \begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{(\langle u,0\rangle,i)}\\ \mathbf{A}^{\mathcal{F}}_{(\langle u,1\rangle,i)}\end{bmatrix}^* \begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}\\ \mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}\end{bmatrix} \\
=& \left((\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{(\langle u,0\rangle,i)}\\ \mathbf{A}_{(\langle u,1\rangle,i)}\end{bmatrix}\right)^* \left((\mathbf{Q}^* \otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{(\langle u,0\rangle,j)}\\ \mathbf{B}_{(\langle u,1\rangle,j)}\end{bmatrix}\right) \\
=& \begin{bmatrix}\mathbf{A}_{(\langle u,0\rangle,i)}\\ \mathbf{A}_{(\langle u,1\rangle,i)}\end{bmatrix}^* \begin{bmatrix}\mathbf{B}_{(\langle u,0\rangle,j)}\\ \mathbf{B}_{(\langle u,1\rangle,j)}\end{bmatrix} \\
=& \mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)} + \mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)}.
\end{aligned}$$

  It is easy to check that $\sum_{u=0}^{p-1}\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}$ is the coefficient of $\omega_q^{k(ip+jpk_A)}$ and $\sum_{u=0}^{p-1}\mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}$ is the coefficient of $\omega_q^{-k(ip+jpk_A)}$. Thus, decoding and summing the corresponding coefficients, allows us to recover $\mathbf{C}_{i,j}$. Note further that the exponent of ω_q is a multiple of p.
- *Case 2: Interference terms.* The terms in (16) with coefficient $\mathbf{A}^{\mathcal{F}*}_{(\langle u,l\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle v,l\rangle,j)}$ with $u \neq v$ are the *interference terms* and they are the coefficients of $\omega_q^{\pm k(ip+u-v+jpk_A)}$. We conclude that the useful terms have no intersection with interference terms since $1 \leq |u-v| < p$.

Next we determine the threshold of the proposed scheme. Towards this end, we find the maximum and minimum degree of $\hat{\mathbf{A}}_k^{\mathcal{F}*}\hat{\mathbf{B}}_k^{\mathcal{F}}$ and then argue that (16) has powers of ω_q that lie at consecutive multiples of k. The threshold can then be obtained by adding 1 to the difference of the maximum and minimum degrees divided by k. The maximum degree of $\hat{\mathbf{A}}_k^{\mathcal{F}*}\hat{\mathbf{B}}_k^{\mathcal{F}}$ is the degree of the term

$$\omega_q^{k(pk_Ak_B-1)}\mathbf{A}^{\mathcal{F}*}_{(\langle p-1,0\rangle,k_A-1)}\mathbf{B}^{\mathcal{F}}_{(\langle 0,0\rangle,k_B-1)},$$

and the minimum degree is the degree of the term

$$\omega_q^{-k(pk_Ak_B-1)}\mathbf{A}^{\mathcal{F}*}_{(\langle p-1,1\rangle,k_A-1)}\hat{\mathbf{B}}^{\mathcal{F}}_{(\langle 0,1\rangle,k_B-1)}.$$

Next we argue that (16) has powers of ω_q that are consecutive multiples of k between the maximum and minimum degree. Towards this end, we show that there always exist some terms in (16) with degree dk, where $-pk_Ak_B+1 \leq d \leq pk_Ak_B-1$. We observe that the positive powers of ${\omega_q}^k$ in (16) can be written as $\pm((j_1-1)p+i_1+1+p-1-i_2+j_2pk_A) = \pm(j_2pk_A + j_1p + i_1 - i_2)$, where $j_1 \in [k_A], j_2 \in [k_B], i_1, i_2 \in [p]$. Consider a positive power $d \leq pk_Ak_B-1$. We can always find a solution such that $j_2 = \lfloor\frac{d}{pk_A}\rfloor$, $j_1 = \lfloor\frac{d \bmod pk_A}{p}\rfloor$, $i_1 - i_2 = (d \bmod pk_A) \bmod p$. A similar result holds when d is negative. We conclude that the threshold of the scheme is $2pk_Ak_B - 1$.

Now suppose that $2pk_Ak_B - 1$ workers return their results. Equation (16) shows that the condition number of the corresponding decoding matrix is equivalent to (up to multiplication by an appropriately defined unitary matrix) a Vandermonde matrix whose parameters are a $(2pk_Ak_B-1)$- sized subset of $\{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$. Therefore, an application of Theorem 1 implies that the worst-case condition number is upper bounded by $O(q^{q-2pk_Ak_B+1+c_1})$.

□

E. Auxiliary Claims

Definition 6 (Permutation Equivalence): We say that a matrix $\mathbf{M}$ is *permutation equivalent* to $\mathbf{M}^\pi$ if $\mathbf{M}^\pi$ can be obtained by permuting the rows and columns of $\mathbf{M}$. We denote this by $\mathbf{M} \asymp \mathbf{M}^\pi$.

Claim 2: Let $\mathbf{M}$ be a $l_1q \times l_2q$ matrix consisting of blocks of size $q \times q$ denoted by $\mathbf{M}_{i,j}$ for $i \in [l_1], j \in [l_2]$ where each $\mathbf{M}_{i,j}$ is a diagonal matrix. Then, the rows and columns of $\mathbf{M}$ can be permuted to obtain $\mathbf{M}^\pi$ which is a block diagonal matrix where each block matrix is of size $l_1 \times l_2$ and there are q of them.

Proof: For an integer a, let $(a)_q$ denote $a \bmod q$. In what follows, we establish two permutations

$$\pi_{l_1}(i) = l_1(i)_q + \lfloor i/q \rfloor, 0 \le i < l_1 q, \text{ and}$$
$$\pi_{l_2}(j) = l_2(j)_q + \lfloor j/q \rfloor, 0 \le j < l_2 q$$

and show that applying row-permutation π_{l_1} and column-permutation π_{l_2} to $\mathbf{M}$ will result in a block diagonal matrix $\mathbf{M}^{\pi}$.

We observe that (i,j)-th entry in $\mathbf{M}$ is the $((i)_q, (j)_q)$-th entry in the block $\mathbf{M}_{\lfloor i/q \rfloor, \lfloor j/q \rfloor}$. Under the applied permutations the (i,j)-th entry in $\mathbf{M}$ is mapped to $(l_1(i)_q + \lfloor i/q \rfloor, l_2(j)_q + \lfloor j/q \rfloor)$-entry in $\mathbf{M}^{\pi}$. Recall that $\mathbf{M}_{\lfloor i/q \rfloor, \lfloor j/q \rfloor}$ is a diagonal matrix which implies that for $(i)_q \neq (j)_q$, the $(l_1(i)_q + \lfloor i/q \rfloor, l_2(j)_q + \lfloor j/q \rfloor)$ entry in $\mathbf{M}^{\pi}$ is 0. Therefore $\mathbf{M}^{\pi}$ is a block diagonal matrix with q blocks of size $l_1 \times l_2$. □

Example 5: Let $l_1 = 2, l_2 = 3, q = 2$. Consider a 4×6 matrix $\mathbf{M}$ which consists of diagonal matrices $\mathbf{M}_{i,j}$ of size 2×2. For $0 \le i \le 1, 0 \le j \le 2$

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{0,0} & \mathbf{M}_{0,1} & \mathbf{M}_{0,2} \\ \mathbf{M}_{1,0} & \mathbf{M}_{1,1} & \mathbf{M}_{1,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & \omega_q & 0 & \omega_q^2 & 0 \\ 0 & 1 & 0 & \omega_q^{-1} & 0 & \omega_q^{-2} \end{bmatrix}.$$

We use row permutation $\pi_{\text{row}} = (0,2,1,3)$, which means $0,1,2,3$-th row of $\mathbf{M}$ permutes to $0,2,1,3$-th row. Similarly, the column permutation is $\pi_{\text{col}} = (0,3,1,4,2,5)$. Thus, $\mathbf{M}^{\pi}$ becomes

$$\mathbf{M}^{\pi} = \begin{bmatrix} 1 & 1 & 1 & & & \\ 1 & \omega_q & \omega_q^2 & & & \\ & & & 1 & 1 & 1 \\ & & & 1 & \omega_q^{-1} & \omega_q^{-2} \end{bmatrix}.$$

Claim 3: (i) Let $a_0(z) = \sum_{j=0}^{\ell_a - 1} a_{j0} z^j$, $a_1(z) = \sum_{j=0}^{\ell_a - 1} a_{j1} z^{-j}$ and $b_0(z) = \sum_{j=0}^{\ell_b - 1} b_{j0} z^{j\ell_a}$, $b_1(z) = \sum_{j=0}^{\ell_b - 1} b_{j1} z^{-j\ell_a}$. Then, $a_{k_1}(z) b_{k_2}(z)$ for $k_1, k_2 = 0,1$ are polynomials that can be recovered from $\ell_a \ell_b$ distinct evaluation points in $\mathbb{C}$.

Let $\mathbf{D}(z^j) = \text{diag}([z^j \ z^{-j}])$ and let

$$\mathbf{X}(z) = \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z) \\ \vdots \\ \mathbf{D}(z^{\ell_a - 1}) \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z^{\ell_a}) \\ \vdots \\ \mathbf{D}(z^{\ell_a(\ell_b - 1)}) \end{bmatrix}.$$

Then, if z_i's are distinct points in $\mathbb{C}$, the matrix

$$[\mathbf{X}(z_1) | \mathbf{X}(z_2) | \ldots | \mathbf{X}(z_{\ell_a \ell_b})],$$

is nonsingular.

(ii) The matrix $[\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \ldots | \mathbf{X}_{i_{\tau-1}}]$ (defined in the proof of Theorem 4) is permutation equivalent to a block-diagonal matrix with four blocks each of size $\tau \times \tau$. Each of these blocks is a Vandermonde matrix with parameters from the set $\{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$.

Proof: First we show that $a_{k_1}(z) b_{k_2}(z)$ for $k_1, k_2 = 0,1$ are polynomials that can be recovered from $\ell_a \ell_b$ distinct evaluation points in $\mathbb{C}$. Towards this end, these four polynomials can be written as

$$a_0(z) b_0(z) = \sum_{i=0}^{\ell_a - 1} \sum_{j=0}^{\ell_b - 1} a_{i0} b_{j0} z^{i + j\ell_a},$$
$$a_0(z) b_1(z) = \sum_{i=0}^{\ell_a - 1} \sum_{j=0}^{\ell_b - 1} a_{i0} b_{j1} z^{i - j\ell_a},$$
$$a_1(z) b_0(z) = \sum_{i=0}^{\ell_a - 1} \sum_{j=0}^{\ell_b - 1} a_{i1} b_{j0} z^{-i + j\ell_a}, \text{ and}$$
$$a_1(z) b_1(z) = \sum_{i=0}^{\ell_a - 1} \sum_{j=0}^{\ell_b - 1} a_{i1} b_{j1} z^{-i - j\ell_a}.$$

Upon inspection, it can be seen that each of the polynomials above has $\ell_a \ell_b$ consecutive powers of z. Therefore, each of these can be interpolated from $\ell_a \ell_b$ non-zero distinct evaluation points in $\mathbb{C}$.

The second part of the claim follows from the above discussion. To see this we note that

$$[a_0(z) \ a_1(z)] = [a_{00} \ a_{01} \ a_{10} \ a_{11} \ \ldots a_{(\ell_a - 1)0} \ a_{(\ell_a - 1)1}] \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z) \\ \vdots \\ \mathbf{D}(z^{\ell_a - 1}) \end{bmatrix} \text{ and}$$

$$[b_0(z) \ b_1(z)] = [b_{00} \ b_{01} \ b_{10} \ b_{11} \ \ldots b_{(\ell_b - 1)0} \ b_{(\ell_b - 1)1}] \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z^{\ell_a}) \\ \vdots \\ \mathbf{D}(z^{\ell_a(\ell_b - 1)}) \end{bmatrix}.$$

Furthermore, the four product polynomials under consideration can be expressed as

$$[a_0(z) \ a_1(z)] \otimes [b_0(z) \ b_1(z)] = \big([a_{00} \ a_{01} \ a_{10} \ a_{11} \ \ldots a_{(\ell_a - 1)0} \ a_{(\ell_a - 1)1}] \otimes [b_{00} \ b_{01} \ b_{10} \ b_{11} \ \ldots b_{(\ell_b - 1)0} \ b_{(\ell_b - 1)1}]\big) \mathbf{X}(z).$$

We have previously shown that all polynomials in $[a_0(z) \ a_1(z)] \otimes [b_0(z) \ b_1(z)]$ can be interpolated by obtaining their values on $\ell_a \ell_b$ non-zero distinct evaluation points. This implies that we can equivalently obtain

$$[a_{00} \ a_{01} \ \ldots a_{(\ell_a - 1)0} \ a_{(\ell_a - 1)1}] \otimes [b_{00} \ b_{01} \ \ldots b_{(\ell_b - 1)0} \ b_{(\ell_b - 1)1}]$$

which means that $[\mathbf{X}(z_1) | \mathbf{X}(z_2) | \ldots | \mathbf{X}(z_{\ell_a \ell_b})]$ is nonsingular. This proves the statement in part (i).

The proof of the statement in (ii) is essentially an exercise in showing the permutation equivalence of several matrices by using Claim 2 and the permutation equivalence properties of Kronecker products. For convenience, we define

$$\mathbf{X}_{l,A} = \begin{bmatrix} \mathbf{I} \\ \Lambda^l \\ \vdots \\ \Lambda^{l(k_A - 1)} \end{bmatrix}, \text{ and}$$

$$\mathbf{X}_l^P = \mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P \asymp \mathbf{X}_l^{P,\pi} = \begin{bmatrix} \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,1} & & & \\ & \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,1} & & \\ & & \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,2} & \\ & & & \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,2} \end{bmatrix}.$$

$$\mathbf{X}_{l,B} = \begin{bmatrix} \mathbf{I} \\ \Lambda^{lk_A} \\ \vdots \\ \Lambda^{lk_A(k_B-1)} \end{bmatrix}$$

so that $\mathbf{X}_l = \mathbf{X}_{l,A} \otimes \mathbf{X}_{l,B}$. Recall that we are analyzing the matrix $\mathbf{X} = [\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\dots|\mathbf{X}_{i_{\tau-1}}]$. An application of Claim 2 shows that (blank entries in the matrices below indicate zero blocks)

$$\mathbf{X}_{l,A} \asymp \mathbf{X}_{l,A}^P = \begin{bmatrix} \mathbf{V}_{l,A,1} & \\ & \mathbf{V}_{l,A,2} \end{bmatrix}, \text{ and}$$
$$\mathbf{X}_{l,B} \asymp \mathbf{X}_{l,B}^P = \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix},$$

where $\mathbf{V}_{l,A,1} = [1, \omega_q^l, \cdots, \omega_q^{l(k_A-1)}]^T$, $\mathbf{V}_{l,A,2} = [1, \omega_q^{-l}, \cdots, \omega_q^{-l(k_A-1)}]^T$. Also, $\mathbf{V}_{l,B,1} = [1, \omega_q^{lk_A}, \cdots, \omega_q^{lk_A(k_B-1)}]^T$, $\mathbf{V}_{l,B,2} = [1, \omega_q^{-lk_A}, \cdots, \omega_q^{-lk_A(k_B-1)}]^T$. Then we conclude that $\mathbf{X} \asymp \mathbf{X}^P = [\mathbf{X}_{i_0}^P|\mathbf{X}_{i_1}^P|\cdots|\mathbf{X}_{i_{\tau-1}}^P]$, where $\mathbf{X}_l^P = \mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P$. The equation at the top of the page shows that $\mathbf{X}_l^P$ is permutation-equivalent to a block-diagonal matrix.

By the definition of Kronecker product, we have

$$\mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P = \begin{bmatrix} \mathbf{V}_{l,A,1} \otimes \mathbf{X}_{l,B}^P & \\ & \mathbf{V}_{l,A,2} \otimes \mathbf{X}_{l,B}^P \end{bmatrix}.$$

Note that $\mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,j} \asymp \mathbf{V}_{l,B,j} \otimes \mathbf{V}_{l,A,i}$, then

$$\begin{aligned} &\mathbf{V}_{l,A,i} \otimes \mathbf{X}_{l,B}^P \\ =&\mathbf{V}_{l,A,i} \otimes \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix} \\ \asymp& \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix} \otimes \mathbf{V}_{l,A,i} \\ =& \begin{bmatrix} \mathbf{V}_{l,B,1} \otimes \mathbf{V}_{l,A,i} & \\ & \mathbf{V}_{l,B,2} \otimes \mathbf{V}_{l,A,i} \end{bmatrix} \\ \asymp& \begin{bmatrix} \mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,2} \end{bmatrix}. \end{aligned}$$

Thus, we can conclude that $\mathbf{X}_l^P \asymp \mathbf{X}_l^{P,\pi}$. In addition, we have

$$\begin{aligned} &\mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,1} \\ &= [1, \omega_q^l, \cdots, \omega_q^{l(k_Ak_B-2)}, \omega_q^{l(k_Ak_B-1)}]^T, \\ &\mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,1} \\ &= [\omega_q^{-l(k_A-1)}, \omega_q^{-l(k_A-2)}, \cdots, \omega_q^{l(k_A(k_B-1)-1)}, \omega_q^{lk_A(k_B-1)}]^T, \\ &\mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,2} \\ &= [\omega_q^{-lk_A(k_B-1)}, \omega_q^{-l(k_A(k_B-1)-1)}, \cdots, \omega_q^{l(k_A-2)}, \omega_q^{l(k_A-1)}]^T, \\ &\text{and } \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,2} \\ &= [\omega_q^{-l(k_Ak_B-1)}, \omega_q^{-l(k_Ak_B-2)}, \cdots, \omega_q^{-l}, 1]^T. \end{aligned}$$

Finally, applying Claim 2 again we obtain the required result. □

Claim 4: Let $\tau_{\text{diff}} = 2k_Ak_Bp - 2(k_Ak_B + pk_A + pk_B) + k_A + k_B + 2p$ where k_A, k_B and p are positive integers with $p > 1$. Then, $\tau_{\text{diff}} < 0$ only if $k_A = 1$ or $k_B = 1$.

Proof: If $k_A = 1$, then $\tau_{\text{diff}} = 1 - k_B < 0$ when $k_B > 1$; a similar argument holds when $k_B = 1, k_A > 1$. On the other hand when $k_A > 1$ and $k_B > 1$, suppose that

$$\begin{aligned} &2k_Ak_Bp + k_A + k_B + 2p < 2(k_Ak_B + pk_A + pk_B), \\ \implies &2 + \frac{1}{k_Bp} + \frac{1}{k_Ap} + \frac{2}{k_Ak_B} < 2\left(\frac{1}{p} + \frac{1}{k_B} + \frac{1}{k_A}\right) \quad (17) \\ &\text{(upon dividing by } k_Ak_Bp). \end{aligned}$$

We note that if k_A, k_B and p are all ≥ 3, then we have a contradiction since the RHS is ≤ 2, whereas the LHS is > 2. Thus, we only need to consider a limited number of cases where some of the values equal 2. These can be verified on a case by case basis. □

REFERENCES

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[2] A. Ramamoorthy, A. B. Das, and L. Tang, "Straggler-resistant distributed matrix computation via coding theory: Removing a bottleneck in large-scale data processing," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 136–145, May 2020.

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

[4] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, Jan. 2019.

[5] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: SIAM, 2002.

[6] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Proc. Syst. (NIPS)*, 2017, pp. 4403–4413.

[7] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Proc. Syst. (NIPS)*, 2016, pp. 2100–2108.

[8] A. E. Yagle, "Fast algorithms for matrix multiplication using pseudo-number-theoretic transforms," *IEEE Trans. Signal Process.*, vol. 43, no. 1, pp. 71–76, Jan. 1995.

[9] V. Y. Pan, "How bad are Vandermonde matrices?" *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 2, pp. 676–694, Jan. 2016.

[10] L. Tang, K. Konstantinidis, and A. Ramamoorthy, "Erasure coding for distributed matrix multiplication for matrices with bounded entries," *IEEE Commun. Lett.*, vol. 23, no. 1, pp. 8–11, Jan. 2019.

[11] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2019, pp. 1777–1781.

[12] A. B. Das, L. Tang, and A. Ramamoorthy, "C^{3}LES: Codes for coded computation that leverage stragglers," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

[13] A. B. Das and A. Ramamoorthy, "Distributed matrix-vector multiplication: A convolutional coding approach," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2019, pp. 3022–3026.

[14] A. B. Das, A. Ramamoorthy, and N. Vaswani, "Efficient and robust distributed matrix computations via convolutional coding," *IEEE Trans. Inf. Theory*, vol. 67, no. 9, pp. 6266–6282, Sep. 2021.
[15] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2758–2785, Jan. 2021.
[16] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random Khatri-Rao-Product codes for numerically-stable distributed matrix multiplication," in *Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2019, pp. 253–259.
[17] A. B. Das and A. Ramamoorthy, "Coded sparse matrix computation schemes that leverage partial stragglers," 2020, *arXiv:2012.06065*.
[18] A. B. Das and A. Ramamoorthy, "A unified treatment of partial stragglers and sparse matrices in coded matrix computation," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Oct. 2021, pp. 1–6.
[19] R. M. Gray, "Toeplitz and circulant matrices: A review," *Found. Trends Commun. Inf. Theory*, vol. 2, no. 3, pp. 155–239, 2006.
[20] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1991.
[21] *Github Repository for Computing Matrix Inverse Over Prime Order Finite Field*. Accessed: May 20, 2020. [Online]. Available: https://github.com/litangsky/inverseoverfield
[22] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–40, 2019.
[23] *Repository of Numerically Stable Coded Matrix Computations Via Circulant and Rotation Matrix Embeddings*. Accessed: Jun. 11, 2020. [Online]. Available: https://github.com/litangsky/stableCodedComputing
[24] J.-P. Berrut and L. N. Trefethen, "Barycentric Lagrange interpolation," *SIAM Rev.*, vol. 46, no. 3, pp. 501–517, 2004.
[25] V. Y. Pan, "TR-2013003: Polynomial evaluation and interpolation: Fast and stable approximate solution," CUNY Acad. Works, 2013. [Online]. Available: https://academicworks.cuny.edu/gc_cs_tr/378

Aditya Ramamoorthy (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi (IIT Delhi), New Delhi, India, and the M.S. and Ph.D. degrees from the University of California, Los Angeles (UCLA).

He is currently the Northrop Grumman Professor of electrical and computer engineering and (by courtesy) of mathematics with Iowa State University, Ames, IA, USA. His research interests include classical/quantum information theory and coding techniques with applications to distributed computation, content distribution networks, and machine learning. He was a recipient of the 2020 Mid-Career Achievement in Research Award, the 2019 Boast-Nilsson Educational Impact Award, and the 2012 Early Career Engineering Faculty Research Award from Iowa State University, the 2012 NSF CAREER Award, and the Harpole-Pentair Professorship in 2009 and 2010. He has served as an Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS from 2011 to 2015 and the IEEE TRANSACTIONS ON INFORMATION THEORY from 2016 to 2019.

Li Tang received the B.E. degree in mechanical engineering and the M.S. degree in electrical and information engineering from Beihang University, Beijing, China, in 2011 and 2014, respectively, and the Ph.D. degree in electrical and computer engineering from Iowa State University, Ames, IA, USA, in 2020. He is currently working as a Research Engineer with Pinterest, Inc., San Francisco, CA, USA. His research interests include coding theory and its application to caching, distributed computation, and machine learning.