Coded Computing for Resilient, Secure, and Privacy-Preserving Distributed Matrix Multiplication

Qian Yu, *Member, IEEE,* A. Salman Avestimehr *Fellow, IEEE*
(*Invited Paper*)

***Abstract*—Coded computing is a new framework to address fundamental issues in large scale distributed computing, by injecting structured randomness and redundancy. We first provide an overview of coded computing and summarize some recent advances. Then we focus on distributed matrix multiplication and consider a common scenario where each worker is assigned a fraction of the multiplication task. In particular, by partitioning two input matrices into m-by-p and p-by-n subblocks, a single multiplication task can be viewed as computing linear combinations of pmn submatrix products, which can be assigned to pmn workers. Such block-partitioning-based designs have been widely studied under the topics of secure, private, and batch computation, where the state of the arts all require computing at least "cubic" (pmn) number of submatrix multiplications. Entangled polynomial codes, first presented for straggler mitigation, provides a powerful method for breaking the cubic barrier. It achieves a subcubic recovery threshold, i.e., recovering the final product from *any* subset of multiplication results with a size order-wise smaller than pmn. We show that entangled polynomial codes can be further extended to also include these three important settings, providing unified frameworks that order-wise reduce the total computational costs by achieving subcubic recovery thresholds.**

***Index Terms*—Distributed Computing, Coding Theory, Matrix Multiplication, Stragglers, Security, Privacy, Batch Processing.**

I. Introduction

Large scale distributed computing faces several modern challenges, in particular, to provide resiliency against stragglers [1], [2], robustness against computing errors [3], security against Byzantine and eavesdropping adversaries [2], [4]–[6], privacy of sensitive information [2], [5]–[7], and to efficiently handle repetitive computation [2]. Coded computing is an emerging field that resolves these issues by introducing and developing new coding theoretic concepts, started focusing on straggler mitigation [8]–[10], then later extended to secure and private computation [2], [7], [11]–[14].

A common approach in distributed computing to handle straggler is to use "uncoded repetition", where the same computation task is duplicated and assigned onto multiple worker machines, to provide the needed resiliency when some of the workers are either slow or fail to return their assigned computation task. Coding for straggler mitigation is first studied in [8] for linear computation, where classical linear codes can be directly applied to achieve the same benefits. Taking matrix-vector multiplication as an example, as illustrated in Figure 1. The goal is to multiply two input matrices of equal sizes, A_0 and A_1, by a globally available vector x, using a set of workers each can compute one matrix-vector multiplication of the same sizes. The conventional approach in distributed computing is to repetitively assign each worker a task of computing either $A_0^\intercal x$ or $A_1^\intercal x$, which requires at least 4 workers to tolerate one straggler. However, utilizing ideas from erasure coding, one can tolerate any single straggler using only 3 worker machines, by first encode the input matrices using a $(3, 2)$-MDS code, then let each worker multiply the assigned coded matrix by x. More generally, it is shown in [8] that one can apply any linear optimal erasure code (essentially linear MDS code) to distributedly compute matrix-vector multiplication, with m input matrices and N worker machines, to tolerate $N - m$ stragglers.

For computation beyond linear functions, new classes of coding designs are needed to achieve optimality. Particularly, consider a general distributed computing structure where the function to be computed at each worker is non-linear (see Section II and Figure 2 for details), a direct application of classical error-correcting codes will not provide guarantees in straggler resiliency, and may even lead

Q. Yu and A. S. Avestimehr are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, 90089 USA (e-mail: qyu880@usc.edu; avestimehr@ee.usc.edu). A part of this paper was presented at ISIT 2020. Manuscript received April 13, 2020; revised August 09, 2020.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2020.3032196, IEEE Transactions on Communications

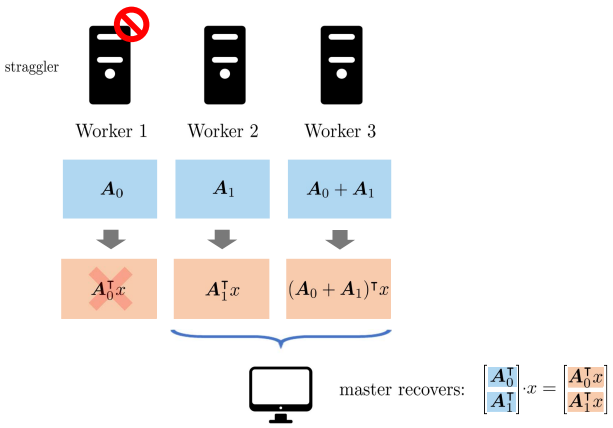

Fig. 1: An illustration of coded computing for linear functions (matrix-vector multiplication). By encoding the input matrices A_0, A_1 using a $(3, 2)$-MDS code, the master can recover both $A_0^\intercal x$ and $A_1^\intercal x$ given results from any 2 out of 3 workers.

to computing results that do not support the recoverability of the needed final output. To understand how to design optimal codes for non-linear computation tasks, in [10] we started by studying a bilinear computation task: matrix-by-matrix multiplication. We introduced the polynomial coded computing (PCC) framework, which is used to developed computing schemes for matrix multiplication, as well as for general computing tasks. We are interested in maximizing the number of stragglers that can be tolerated while fixing the storage and computation loads at each worker. This is equivalent to minimize a quantity called the *recovery threshold* [10], defined as the number of workers the master has to wait for in the worst case to guarantees the recoverability of the final result. The main idea of PCC is to jointly encode the input variables into single variate polynomials where the coefficients are functions of the inputs. By assigning each worker evaluations of these polynomials as coded variables, they essentially evaluate a new polynomial composed by each worker's computation and the encoding functions at the same point. As long as the needed final results can be recovered from the coefficients of the composed polynomial, the master can decode the final output when sufficiently many workers complete their computation. PCC significantly reduces the design problem of coded computation to finding polynomials satisfying the above decodability constraint while minimizing the degree of the decomposed polynomial. It has been shown that PCC achieves great success in providing exact optimal coding constructions for large classes of computation tasks including pair-wise product [10], convolution [10], [15], inner product [15], [16], element-wise product [15], and general batch multivariate-polynomial evaluation [2].

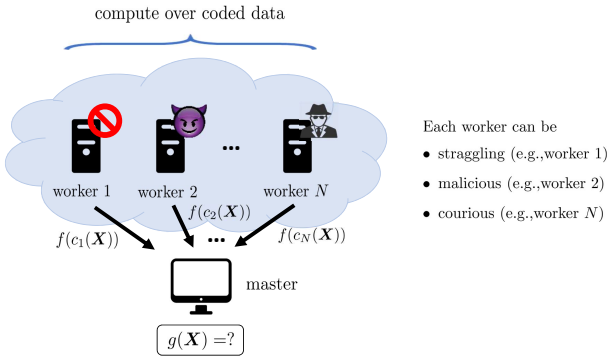

Fig. 2: An illustration of a general coded computation problem. A collection of workers aim to compute a function g given an input dataset, where each worker can return an evaluation of a function f with possibly coded data assignments. By carefully designing the coding functions (c_i's), the final results can be efficiently recovered after computation is applies on coded data, in the presence of stragglers, while providing security and privacy against malicious and colluding workers.

An important problem in distributed matrix multiplication is to consider the case where the inputs are encoded and multiplied in a block-wise manner. This setup generalizes the problem formulated in [10] to enable a more flexible tradeoff between resources such as storage, computation, and communication. It has been studied in [15]–[21]. For straggler mitigation, the state of the art is achieved by two versions of the entangled polynomial code, both first presented in [15], which characterizes the optimum recovery threshold within a factor of 2. For brevity, we refer to the collection of them as *entangled polynomial codes*. One significance of entangled polynomial codes is that it maps non-straggler-mitigating linear coded computing schemes to bilinear-complexity decompositions, which bridges the areas of fast matrix multiplication and coded computation, enables utilizing techniques developed in the rich literature (e.g., [22]–[41]). Moreover, this connection reduces block-wise matrix multiplication to computing element-wise products, for which we developed the optimal strategy

for straggler mitigation.

Besides straggler mitigation, coded computing has been extended to include two other directions. One is to provide fault tolerance against computing errors, where a number of workers can produce unpredicted faulty results and the goal is to minimize the number of errors that can be detected or corrected. The other is to provide security against Byzantine adversaries that can intentionally manipulate their computing results, and to tolerate as many of them as possible. The coding gain achieved by entangled polynomial codes extended to fault-tolerant computing [42] and it is shown in [2] that security against Byzantine adversaries can also be provided the same way.

In this paper, we first provide an overview of coded computation and summarize some of the earlier results. Then we extend entangled polynomial codes to three main problems: secure, private, and batch distributed matrix multiplication. In secure distributed matrix multiplication [11]–[14], [17]–[19], [43]–[50], the goal is to compute a single matrix product while preserving the privacy of input matrices against eavesdropping adversaries (Fig. 6); in private distributed matrix multiplication [13], [14], [18], [51], the goal is to multiply a single pair from two lists of matrices while keeping the request (indices) private (Fig. 7); batch distributed matrix multiplication [20], [21], [48] considers a scenario where more than one pair of matrices are to be multiplied (Fig. 8). In all these cases, we aim to find computing schemes to minimizes the recovery threshold, as well as characterizing the minimum number of workers required for a valid design.

There are recent works on each of these problems that considered general block-wise partitioning of input matrices [17]–[21]. However, all results presented in prior works are limited by a "cubic" barrier. Explicitly, when the input matrices to be multiplied are partitioned into m-by-p and p-by-n subblocks, all state of the arts require the workers computing at least pmn products of coded submatrices per each multiplication task.

We demonstrate how entangled polynomial codes can be extended to break the cubic barrier in all three problems. We show that the coding ideas of entangled polynomial codes and PCC can be applied to provide unified solutions with needed security and privacy, as well as efficiently handling batch evaluation. Moreover, we achieve order-wise improvements upon state-of-the-art designs with explicit coding constructions.

II. Overview of Coded Computation

A main challenge in distributed computing is to design schemes to operate in the presence of stragglers. Coded computation has been proposed as an effective approach to mitigate the straggler effect, and computing strategies has been proposed for a variety of computation tasks, including matrix multiplication [8]–[10], [15], [16], convolution [10], [52], Fourier transformation [53], [54], element-wise multiplication [15], and multivariate-polynomial evaluation [2]. The main idea of coded computing is to assign each worker data or tasks in carefully designed coded forms, such that the final result can still be recovered after possibly non-linear computation is applied on coded data. In the rest of this section, we present a general formulation for coded computing problems and illustrate with a few examples.

A. General Formulation

In a standard framework of coded computation (illustrated in Figure 2), we aim to design a computing scheme to compute a function g over an input dataset $\boldsymbol{X}$ using N distributed workers. Each worker computes a single evaluation of some function f, which can be viewed as building blocks of computing g. A conventional approach in distributed computation is to assign each worker an *uncoded* fraction of the input dataset and to recover the final results from evaluations of these uncoded assignments.

A coded computing design that uses N workers first encodes the dataset $\boldsymbol{X}$ using N encoding functions $\boldsymbol{c} \triangleq (c_1, ..., c_N)$, then assign $c_i(\boldsymbol{X})$ to each worker i as the coded input. In the presence of stragglers, the decoder waits for a subset of fastest workers until $g(\boldsymbol{X})$ can be recovered given the returned results from the workers using some decoding functions.

We say a coded computing scheme achieves a *recovery threshold* of R, if the master can correctly decode the final output given the computing results from *any* subset of at least R workers. This is an equivalent measure of the number of stragglers (as well as the number of Byzantine adversaries) that can be tolerated. The goal is to design the encoding

and decoding functions to minimize the achievable recovery threshold given f, g, and N.

More rigorously, let $\boldsymbol{d}$ denote the collection of decoding functions used by the master over all possible subsets of workers that return their assigned computation. A coded computing scheme, essentially described by the pair $(\boldsymbol{c}, \boldsymbol{d})$, is to be designed under practical constraints. We denoted the set of allowable constructions by $\mathcal{S}$. For example, an important constraint is to ensure the complexities of the encoding and decoding functions are reasonably low, which is guaranteed in most related works by focusing on linear codes. In certain applications, the privacy of the input dataset is required, which can also be written as a constraint on encoding functions.

In a coded computing problem, parameterized by $(f, g, N, \mathcal{S})$, we aim to characterize the minimum recovery threshold, denoted K^*, defined as

$$K^* \triangleq \min_{(\boldsymbol{c},\boldsymbol{d})\in\mathcal{S}} K_{f,g,N}(\boldsymbol{c}, \boldsymbol{d}), \tag{1}$$

where $K_{f,g,N}(\boldsymbol{c}, \boldsymbol{d})$ denoted the recovery threshold achieved by $(\boldsymbol{c}, \boldsymbol{d})$, as well as finding computing schemes that achieve K^* as close as possible.

Remark 1. Besides the recovery threshold, there are several other important performance metrics, including the storage, computation, and communication costs at each worker. All these quantities are simply characterized by function f. Hence, it is possible to focus on minimizing the recovery threshold without worrying about significantly increasing the costs in those aspects. Explicitly, these quantities are characterized by

- Storage cost $= O$(input size of f)
- Computation cost $= O$(computational complexity of f)
- Communication cost $= O$(output size of f)

When the encoding functions are linear (e.g., as defined in [2], [15]), the above characterizations are tight. In practice, there could be an algorithmic degree of freedom in choosing between different f to compute the same function g. By finding the optimal recovery thresholds for each f, one can optimally switch between different algorithmic designs and tradeoff resources such as storage, computation, and communication. However, in many scenarios, the identity of f that leads to the best overall performance depends on the computing environment and needs to be found empirically.

Polynomial Coded Computing (PCC). A key challenge in coded computing is to design coding functions when f and g are nonlinear, to still ensure efficient recovery of final results after nonlinear computation is applied on coded variables. The PCC framework was introduced in [10], which achieves optimal recovery thresholds for large classes of functions including matrix multiplication, convolution, and polynomial evaluation [2], [10], [15]. A general PCC design encodes the input dataset by assigning the workers evaluations of a carefully designed single variate polynomial. More specifically, the coding design is based on the following design parameters:

- A single variate polynomial $c(\cdot)$, where the coefficients are possibly random functions of the input variables.
- N evaluation points denoted $y_1, ..., y_N$ from the base field.

Then each worker i obtains $c_i(\boldsymbol{X}) = c(y_i)$ as the encoded variable.

After the workers apply f on their assignments, they essentially evaluate the composed polynomial $f(c)$ at the same points. Hence, if the evaluation points $y_1, ..., y_N$ are distinct, and the decoder receives results from sufficiently many (at least the degree of $f(c)$ plus one) workers, they can recover all information about polynomial $f(c)$. Based on this observation, the design problem in the PCC framework is to construct a polynomial c, satisfying[1]

- Decodability: the final result can be computed using coefficients of $f(c)$,

while minimizing the degree of $f(c)$ to achieve minimum recovery thresholds.

PCC provides several other properties of interests: linearly-coded constructions of polynomial c lead to linear codes; the encoding/decoding costs due to polynomial evaluation and interpolation can be handled using efficient algorithms with almost linear complexities [55]. We present three example problems, starting with linear computation, then demonstrate how PCC is applied to construct optimal codes for non-linear computation tasks.

[1]As well as other possible requirements such as complexity constraints on encoding and decoding functions (e.g., linear codes) [15], and data-privacy [2].

B. Matrix-vector Multiplication

The matrix-vector multiplication problem we mentioned earlier in the introduction can be formulated as a coded computing problem as follows. Given any fixed vector $x \in \mathbb{F}^s$ and an input matrix $A \in \mathbb{F}^{s\times t}$, the goal is to compute $g(A) = A^\intercal x$. Each worker i can process a smaller coded version of the input, with size $c_i(A) \in \mathbb{F}^{s\times \frac{t}{m}}$, and compute a function f specified by $f(c_i(A)) = c_i(A)^\intercal x$. Let A be column-wise partitioned into m submatrices $A_0, ..., A_{m-1}$ of equal sizes, we are interested in a class of computing designs (i.e., the set $\mathcal{S}$), where each encoding function computes a linear combination of these submatrices. It is shown in [8] that this matrix-vector multiplication problem reduces to design classical erasure codes, where the class of solution that tolerates the maximum number of erasure errors[2] is referred to as MDS codes. By applying any linear MDS code to encode the input submatrices, one can achieve a recovery threshold of m.

C. Matrix-matrix Multiplication

Consider a scenario where the goal is to compute the product $A^\intercal B$ given two large matrices $A \in \mathbb{F}^{s\times t}$ and $B \in \mathbb{F}^{s\times r}$. Here the input dataset is $X = (A, B)$, and the computation task is $g(A, B) = A^\intercal B$. After partitioning the input matrices column-wise into m and n submatrices of equal sizes, denoted $A_0, ..., A_{m-1}$ and $B_0, ..., B_{n-1}$, the final results are essentially the collection of all mn pair-wise submatrix-products $A_i^\intercal B_j$'s. If each worker can compute a single submatrix product of the same sizes, i.e., f is the multiplication of two matrices of sizes $\mathbb{F}^{\frac{t}{m}\times s}$ and $\mathbb{F}^{s\times \frac{r}{n}}$, an uncoded design using $K = mn$ workers can be constructed by assigning the workers distinct (A_i, B_j)'s as inputs.

Polynomial codes was proposed in [10], which encodes the input dataset using the following polynomial: $c(x) = (\sum_{j=0}^{m-1} A_j x^j, \sum_{j'=0}^{n-1} B_{j'} x^{j'm})$. More explicitly, given any distinct evaluation points $y_1, ..., y_N$, each worker i obtains $(\tilde{A}_i, \tilde{B}_i)$ where

$$\tilde{A}_i = \sum_{j=0}^{m-1} A_j y_i^j, \qquad \tilde{B}_i = \sum_{j'=0}^{n-1} B_{j'} y_i^{j'm}. \tag{2}$$

After the workers multiply the assigned coded matrices, they essentially evaluate the composed polynomial $f(c(x)) = \sum_{j=0}^{m-1}\sum_{j'=0}^{n-1} A_j^\intercal B_{j'} x^{j+j'm}$, which has a degree of $mn-1$ and the mn coefficients are exactly the mn needed submatrix-products. After computation results are received from *any* subset of mn workers, the final results can be recovered from polynomial interpolation. I.e., achieving a recovery threshold of mn. It is proved in [10] that polynomial codes achieve the optimal recovery threshold for this scenario. Polynomial codes were later extended to allow for general block-wise partitioning of the input matrices, as discussed in Section III.

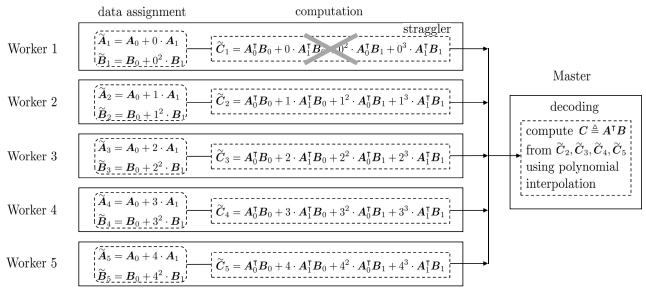

Fig. 3: An illustration of polynomial code for matrix multiplication using 5 workers that can each store half of each input matrix. The final result can be decoded from *any* 4 workers.

D. Polynomial Computation

Another example is to evaluate multivariate polynomials on a dataset $X = (X_1, ..., X_K)$. Explicitly, given a general polynomial f, the goal is to compute $g(\boldsymbol{X}) = (f(X_1), ..., f(X_K))$. If each worker can compute a single evaluation, then an uncoded design using K workers can be obtained by assigning each worker i input X_i.

For straggler mitigation, Lagrange Coded Computing (LCC) was proposed in [10], which encodes the input variables using the Lagrange polynomial $c(x) \triangleq \sum_{j\in[K]} X_j \cdot \prod_{k\in[K]\setminus\{j\}} \frac{x-x_k}{x_j-x_k}$ where $x_1, ..., x_K$ are some arbitrary distinct elements from the base field $\mathbb{F}$. In other words, each worker i obtains the following $\tilde{X}_i$ as the coded variable.

$$\tilde{X}_i \triangleq \sum_{j\in[K]} X_j \cdot \prod_{k\in[K]\setminus\{j\}} \frac{y_i - x_k}{x_j - x_k}. \tag{3}$$

After each worker applies function f over the coded inputs, they essentially evaluate the composed polynomial $f(c)$, of which the evaluations at $x_1,...,x_K$ are exactly the K needed final results.

[2]When the based field is sufficiently large.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCOMM.2020.3032196, IEEE Transactions on Communications

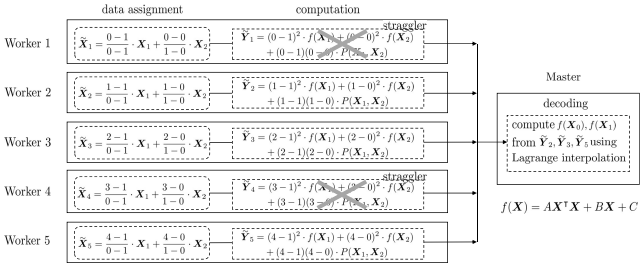

Fig. 4: An illustration of LCC for evaluating a general degree-2 multivariate polynomial $f(X) = AX^\intercal X + BX + C$ using 5 workers with an input dataset $\boldsymbol{X} = (X_1, X_2)$ that consists of two square matrices. The final result can be decoded from *any* 3 workers.

Let $\deg f$ denote the total degree of polynomial f, the degree of the composed polynomial equals $(K-1)\deg f$. By assigning each worker distinct evaluation points, the decoder can recover all final results by receiving results from any subset of $(K-1)\deg f+1$. This exactly achieves the optimum recovery thresholds among all linear codes when the number of workers N is sufficiently large, while in other cases the optimum recovery thresholds are achieved by an uncoded version of LCC, where the evaluation points $y_1, ..., y_N$ are selected from $x_1, ..., x_K$ [2]. It was shown in [2] that by padding the input dataset with random keys, LCC simultaneously provides security against malicious workers and privacy of data against colluding workers, and achieves the optimal tradeoff between straggler resiliency, security, and privacy.

III. Coded Computing for Block-Partitioned Matrix Multiplication

The main focus of this paper is to consider a more generalized matrix multiplication setting where the inputs are block-wise partitioned. In a basic setup, given a pair of input matrices $A \in \mathbb{F}^{s\times t}$, $B \in \mathbb{F}^{s\times r}$ for a sufficiently large field $\mathbb{F}$, each worker i is assigned a pair of possibly coded matrices $\tilde{A}_i \in \mathbb{F}^{\frac{s}{p}\times\frac{t}{m}}$ and $\tilde{B}_i \in \mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$, which are encoded based on some (possibly random) functions of the input matrices respectively (see Figure 5). The workers can each compute $\tilde{C}_i \triangleq \tilde{A}_i^\intercal \tilde{B}_i$ and return them to the master. The master tries to recover the final product $C \triangleq A^\intercal B$ based on results from a subset of fastest workers using some decoding functions. In another word, we are considering a coded computing problem where g is a multiplication of two matrices of sizes $\mathbb{F}^{t\times s}$ and $\mathbb{F}^{s\times r}$, and f is a multiplication of two matrices of sizes $\mathbb{F}^{\frac{t}{m}\times\frac{s}{p}}$ and $\mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$. By partitioning the input matrices into p-by-m and p-by-n subblocks, the product of two input matrices can thus be viewed as linear combinations of pmn submatrix products according to block-matrix-multiplication rules, which can be computed using pmn workers with uncoded inputs. We aim to achieve the minimum possible recovery threshold given parameters $p, m, n,$ and N.

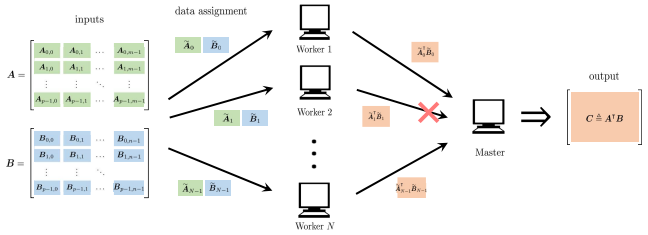

Fig. 5: Overview of the distributed matrix multiplication problem with general block-partitioning of the inputs. Each worker is assigned two possibly coded submatrices and computes their product. The master aims to decode the product of the input matrices based on results from non-straggling workers.

The best-known recovery threshold for computing block-partitioned matrix multiplication is achieved by a class of PCC designs referred to as entangled polynomial codes. In particular, entangled polynomial codes achieve a recovery threshold of $\min\{pmn+p-1, 2R(p,m,n)-1\}$ for any $p, m,$ and n [15]. Here $R(p,m,n)$ denotes the bilinear complexity [56] for multiplying two matrices of sizes m-by-p and p-by-n. It is well known that $R(p,m,n)$ is subcubic, i.e, $R(p,m,n) = o(pmn)$ when p, m, and n are large. Hence, it order-wise outperforms other block-partitioning-based schemes in related works for straggler mitigation (e.g., [16]).

Remark 2. The quantity bilinear complexity $R(p,m,n)$ should not be confused with a closely related concept: the computational complexity of matrix multiplication. The computational complexity captures the costs from all operations to compute a function. As the relative costs for each basic operation could be variable in different computing systems, the computational complexity is often stated in an inexplicit form (using big-O notation) and most related works focus on studying its asymptotic behaviour. On the other hand, the bilinear com-

plexity $R(p,m,n)$ is a well-defined integer given any p,m,n and the base field $\mathbb{F}$, which can be accurately stated and characterized. For example[3] $R(2,2,2)=7$, which indicates that even for a basic scenario where the inputs are partitioned into 2-by-2 subblocks and no straggler mitigation is required, one should consider using linear codes to reduce the number of workers from 8 to 7, as long as the input matrices are large enough such that coding overheads are negligible [22], [42].

We present entangled polynomial codes as follows. The input matrices are partitioned into p-by-m and p-by-n submatrices of equal sizes, denoted $A_{j,k}$ and $B_{j,k'}$ for $j \in \{0,...,p-1\}$, $k \in \{0,...,m-1\}$, and $k' \in \{0,...,n-1\}$, and we aim to recover $C_{k,k'} \triangleq \sum_i A_{j,k}^{\intercal} B_{j,k'}$ for any k and k'. A basic version of entangled polynomial codes is designed to achieve a recovery threshold of $pmn+p-1$, which generalizes the Polynomial codes, and encodes the input variables using $c(x) = (\sum_{j=0}^{p-1}\sum_{k=0}^{m-1} A_{j,k}x^{j+kp}, \sum_{j=0}^{p-1}\sum_{k'=0}^{n-1} B_{j,k'}x^{p-1-j+k'pm})$. Explicitly, given any distinct evaluation points $y_1,...,y_N$, each worker i obtains $(\tilde{A}_i,\tilde{B}_i)$ where

$$\tilde{A}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1} A_{j,k}y_i^{j+kp},$$
$$\tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k'=0}^{n-1} B_{j,k'}y_i^{p-1-j+k'pm}. \quad (4)$$

This coding construction results in the following composed polynomial

$$f(c(x)) = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1}\sum_{j'=0}^{p-1}\sum_{k'=0}^{n-1} A_{j,k}^{\intercal}B_{j',k'}x^{(p-1+j-j')+kp+k'pm}, \quad (5)$$

which has a degree of $pmn+p-2$, and the mn needed linear combinations are exactly provided by mn of its coefficients.

Importantly, in the same paper, an improved version of the entangled polynomial code is presented to approach the optimal recovery threshold for general p,m, and n, which achieves a recovery threshold of $2R(p,m,n)-1$ (Theorem 3, [15]). Given any upper bound construction of $R(p,m,n)$ (e.g., Strassen's construction) with rank R and tensor tuples $a \in \mathbb{F}^{R\times p\times m}$, $b \in \mathbb{F}^{R\times p\times n}$, and $c \in \mathbb{F}^{R\times m\times n}$, the inputs are each pre-encoded into a list of R coded submatrices.[4]

$$\tilde{A}_{i,\text{vec}} \triangleq \sum_{j,k} A_{j,k}a_{ijk}, \qquad \tilde{B}_{i,\text{vec}} \triangleq \sum_{j,k} B_{j,k}b_{ijk}. \quad (6)$$

Then the variables are encoded using $c(x) = \sum_j (\tilde{A}_{j,\text{vec}},\tilde{B}_{j,\text{vec}}) \cdot \prod_{k\neq j}\frac{(x-x_k)}{(x_j-x_k)}$. I.e., each worker obtains

$$\tilde{A}_i = \sum_j \tilde{A}_{j,\text{vec}} \cdot \prod_{k\neq j}\frac{(y_i-x_k)}{(x_j-x_k)}, \quad (7)$$
$$\tilde{B}_i = \sum_j \tilde{B}_{j,\text{vec}} \cdot \prod_{k\neq j}\frac{(y_i-x_k)}{(x_j-x_k)}, \quad (8)$$

where x_1,...,x_R are arbitrary distinct elements of $\mathbb{F}$. The coding construction provides a composed polynomial $f(c)$ with degree $2R-2$, which could achieve $2R(p,m,n)-2$ by using upper bound constructions with a rank of $R(p,m,n)$. The final results can be decoded by re-evaluating the composed polynomial at points $x_1,...,x_R$, then combining them based on tensor c.

Note that even for cases where $R(p,m,n)$ is not yet known, one can still obtain explicit coding constructions by swapping in any upper bound constructions (e.g, [23]–[25], [27]–[32], [34]–[36], [38]–[40]). Subcubic recovery thresholds can still be achieved for any sufficiently large p, m, and n even one only applies the well known Strassen's construction [23]. Hence, for simplicity, in this work, we present all results in terms of $R(p,m,n)$, and explicit subcubic constructions can be obtained in the same way.

We focus on linear codes, defined similarly as in [2], [42], which guarantees linear coding complexities w.r.t. the sizes of input matrices, and are dimension independent. Precisely, in a linear coding design, the input matrix A (or each input A for more general settings) is partitioned into p-by-m subblocks of equals sizes (and possibly padded with a list of i.i.d. uniformly random matrices of same sizes, referred to as random keys).[5] Matrix (or matrices) B are partitioned similarly. Each worker is then assigned a pair of linear combinations of these two lists of submatrices as coded inputs. Moreover,

[3]More generally, $R(p,m,n) < pmn$ for any $p,m,n > 1$.

[4]For detailed definitions, see [42].

[5]To make sure the setting is well defined, we assume $\mathbb{F}$ is finite whenever data-security or privacy is taken into account.

the master uses decoding functions that compute linear combinations of received computing results.[6]

All results presented in this paper for distributed matrix multiplication directly extends to general codes with possibly non-linear constructions, by swapping any upper bound of $R(p,m,n)$ into the number workers required by any computing scheme, as we illustrated in [42].

IV. Coded computing for secure and privacy-preserving matrix multiplication

Distributed matrix multiplication is well studied in the context of straggler mitigation. Our goal in this work is to leverage entangled polynomial codes to the settings of secure, private, and batch distributed matrix multiplication, achieving order-wise improvement with subcubic recovery thresholds while meeting the systems' requirements.

A. Secure Distributed Matrix Multiplication

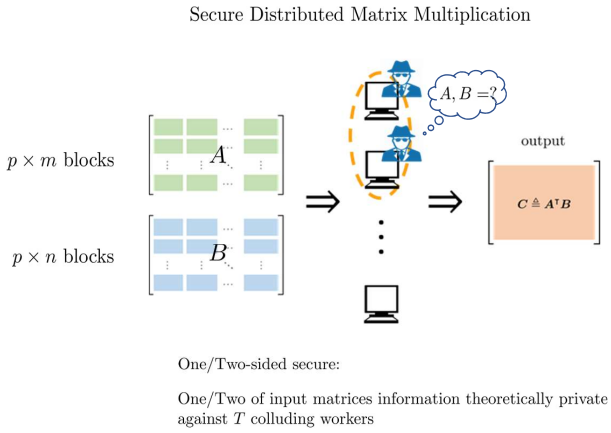

Fig. 6: An illustration of secure distributed matrix multiplication.

Secure distributed matrix multiplication follows a similar setup discussed in Section II, where the goal is to multiply a single pair of matrices, with additional constraints that either one or both of the input matrices are information-theoretic private to the workers, even if up to a certain number of them can collude. In particular,

[6]Note that by relaxing certain assumptions made in the paper, such as allowing the decoder to access inputs and random keys and allowing extra computational cost at workers or master, one can further reduce the recovery threshold (e.g., see discussions in [48], [57]).

Definition 1. *An encoding scheme is one-sided T-secure, if*

$$I(\{\tilde{A}_i, \tilde{B}_i\}_{i\in\mathcal{T}}; A) = 0 \tag{9}$$

for any subset $\mathcal{T}$ with a size of at most T, where A is generated uniformly at random.

Definition 2. *An encoding scheme is fully T-secure, if*

$$I(\{\tilde{A}_i, \tilde{B}_i\}_{i\in\mathcal{T}}; A, B) = 0 \tag{10}$$

is satisfied for any $|\mathcal{T}| \leq T$, for uniformly randomly generated A and B.

Secure distributed matrix multiplication has been studied in [11]–[14], [17]–[19], [43]–[50]. In particular, [17]–[19] presented coded computing designs for general block-wise partitioning of the input matrices, all requiring at least pmn workers' computation.[7] Entangled polynomial codes achieve subcubic recovery thresholds for both one-sided and fully secure settings, formally stated in the following theorem.

Theorem 1. *For secure distributed matrix multiplication, there are one-sided T-secure linear coding schemes that achieve a recovery threshold of $2R(p,m,n) + T - 1$, and fully T-secure linear coding schemes that achieve a recovery threshold of $2R(p,m,n) + 2T - 1$.*

Remark 3. Entangled polynomial codes order-wise improve the state of the arts for general block-wise partitioning [17]–[19], by providing explicit constructions that require subcubic number of workers. In particular, the fully T-secure codes presented in [17]–[19] all require using at least $pmn + 2T$ workers, while the fully T-secure entangled polynomial codes only require at most $O(\frac{pmn}{\min\{p,m,n\}^{0.19}}) + 2T$ workers according to Strassen's upper bound, which is order-wise smaller for any large p, m, and n. Similarly, entangled polynomial codes require order-wise smaller number of workers compared to the one-sided 1-secure coding scheme proposed in [18] (which also satisfies a privacy requirement, to be discussed in the next subsection). Moreover, entangled polynomial codes simultaneously handle data security and straggler issues by tolerating arbitrarily many stragglers while maintaining the same recovery threshold and privacy level.

[7]In addition, at least T extra workers are needed per each input matrix required to be stored securely.

Remark 4. Entangled polynomial codes enables breaking the "cubic" barrier when resiliency, security, or privacy is required, by providing a class of codes that operates upon any general coding structures. It supports codes developed based on ideas from matrix multiplication algorithms (which does not naturally provide resiliency or security[8]) and injects tailored coding designs to allow achieving similar sub-cubic recovery properties.

Remark 5. Following similar converse proof steps we developed in [2], [42], one can show that any linear code that is either one-sided T-secure or fully T-secure requires using at least $R(p,m,n)+T$ workers.[9] Hence, entangled polynomial codes enable achieving optimal recovery thresholds within a factor of 2 for both settings.

B. Private Distributed Matrix Multiplication

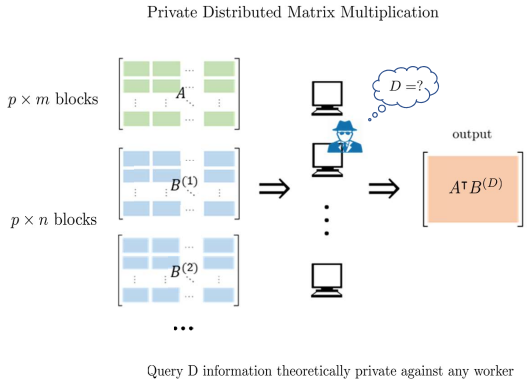

Fig. 7: An illustration of private distributed matrix multiplication.

Private distributed matrix multiplication has been studied in [13], [14], [18], [51], where the goal is to instead multiply a matrix A by one of the matrices $B^{(D)}$ from $\boldsymbol{B}=(B^{(1)},...,B^{(M)})$ while keeping the request D private to the workers. In particular, the master sends a (possibly random) query Q_i to each worker i based the request D. Then the matrices $\boldsymbol{B}$ are encoded by each worker i into a coded submatrix $\tilde{B}_i \in \mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$ based on Q_i. The matrix A is encoded the same as the basic setting, and each worker computes the product of their coded matrices.

The index D should be kept private to any single worker. Precisely,[10]

Definition 3. *In private distributed matrix multiplication, we say a computing scheme is private, if*

$$I(D;Q_i,\tilde{A}_i,\boldsymbol{B})=0 \tag{11}$$

for any $i\in[N]$, where $A,\boldsymbol{B},D$ are sampled uniformly at random.

The master decodes the final output based on the returned results, the request D, and query Q_i's.

Moreover, in some related works [13], [14], [18], the encoding of A is also required to be secure against any single curious worker. I.e.,

Definition 4. *A computing scheme is private and secure, if it is private, and*

$$I(\tilde{A}_i;A)=0 \tag{12}$$

for any $i\in[N]$ for A sampled uniformly randomly.

This setting is referred to as private and secure distributed matrix multiplication. The state of the art for private and secure distributed matrix multiplication with general block-partitioning based designs was proposed in [18], which requires at least pmn number of workers. Entangled polynomial codes achieve subcubic recovery thresholds, formally stated in the following theorem.

Theorem 2. *For private coded matrix multiplication, there are linear coding schemes that achieve a recovery threshold of $2R(p,m,n)$. For private and secure distributed matrix multiplication, linear coding schemes can achieve a recovery threshold of $2R(p,m,n)+1$.*

Remark 6. Entangled polynomial codes order-wise improve the state of the arts for general block-wise partitioning [18], by providing explicit constructions that achieve subcubic recovery thresholds, while simultaneously provides straggler-resiliency, data-security and privacy. As mentioned in Remark 3, [18] presents a private and secure matrix multiplication design that requires at least pmn workers, while entangled polynomial codes provide a private and secure design that requires at most $O(\frac{pmn}{\min\{p,m,n\}^{0.19}})$ workers according to Strassen's upper bound, which is order-wise smaller for any large p,m, and n.

[8]For example, Strassen's construction [22] leads to a computing design with 7 workers for the most basic setting [15], however, one can not always achieve the same recovery threshold for the same matrix partitioning even for tolerating 1 straggler.

[9]A proof is provided in Appendix A.

[10]Note that a stronger privacy condition $I(D;Q_i,\tilde{A}_i,A,\boldsymbol{B})=0$ can still be achieved, if one uses the scheme for private and secure distributed matrix multiplication presented later in this paper.

Remark 7. Similar to the discussion in Remark 5, one can show that any linear code requires at least $R(p,m,n)$ workers for private coded matrix multiplication and $R(p,m,n)+1$ workers for private and secure distributed matrix multiplication, even if one ignores the privacy requirement.[11] This indicates a factor-of-2 optimality of entangled polynomial codes for both settings.

Entangled polynomial codes also apply to a more general scenario where the encoding functions for both input matrices are assigned to the workers, which we refer to as *fully private coded matrix multiplication* and formulate as follows. In fully private coded matrix multiplication, we have two lists of input matrices $\boldsymbol{A} = (A^{(1)},...,A^{(M)})$ and $\boldsymbol{B} = (B^{(1)},...,B^{(M)})$, and the master aims to compute $A^{(D)\intercal}B^{(D)}$ given an index D. We assume $M > 1$, because otherwise the privacy requirement is trivial.

We aim to find computation designs such that D is private against any single worker. Explicitly, the master sends a (possibly random) query Q_i to each worker i based on the demand D. Then worker i encodes *both* A and B based on Q_i. We require the requests to be private in the following sense.

Definition 5. *A computing scheme is fully private, if*

$$I(D; Q_i, \boldsymbol{A}, \boldsymbol{B}) = 0 \tag{13}$$

for any $i \in [N]$*, where* $\boldsymbol{A}, \boldsymbol{B}, D$ *are sampled uniformly at random.*

We summarize the performance of entangled polynomial codes for fully private coded matrix multiplication as follows.

Theorem 3. *For fully private coded matrix multiplication, there are linear coding schemes that achieve a recovery threshold of* $2R(p,m,n)+1$.

Remark 8. Similar to earlier discussions, entangled polynomial codes provide coding constructions for fully private coded matrix multiplication with subcubic recovery thresholds. One can prove that any fully private linear code requires at least $R(p,m,n)+1$ workers.[12] Hence, the factor-of-2 optimality of entangled polynomial codes also holds true for fully private coded matrix multiplication.

[11] A proof is provided in Appendix A.

[12] A proof is provided in Appendix A.

C. Batch Distributed Matrix Multiplication

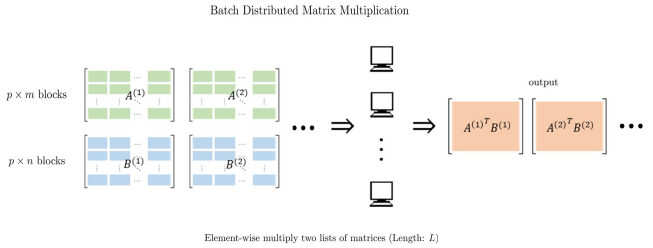

Fig. 8: An illustration of batch distributed matrix multiplication.

The authors of [20], [21], [48] considered a scenario where the goal is to compute L copies of the matrix multiplication task in one round of communication. Formally, a basic setting for batch distributed matrix multiplication is that we have two lists of input matrices $\boldsymbol{A} = (A^{(1)},...,A^{(L)})$ and $\boldsymbol{B} = (B^{(1)},...,B^{(L)})$, and the master aims to compute their element-wise product $\boldsymbol{C} = (A^{(1)\intercal}B^{(1)},...,A^{(L)\intercal}B^{(L)})$. Given partitioning parameters $p, m,$ and n, each worker still computes a single multiplication of coded submatrices with sizes $\mathbb{F}^{\frac{t}{m}\times\frac{s}{p}}$ and $\mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$.

For general block-partitioning based schemes, the state of the art design is provided in [20], [21], where the focus is to reduce the recovery threshold and no security or privacy is required. All known coding constructions presented for batch distributed matrix multiplication requires cubic number of workers per each multiplication task even no straggler presents (i.e., requiring at least $Lpmn$ workers in total).

We show that entangled polynomial codes offer a unified coding framework for batch matrix multiplication, achieving subcubic recovery thresholds while simultaneously handling all security and privacy requirements that are discussed earlier in this section. We present this result in the following theorem.[13] The proofs and detailed formulations can be found in Section VII.

Theorem 4. *For coded distributed batch matrix multiplication with parameters* $p, m, n,$ *and* L*, there*

[13] Similar to [42], in the most basic scenario with no requirements on resiliency, security, and privacy (i.e., requiring a recovery threshold of N, with $T = 0$ and $M = 1$), one can directly apply any upper bound construction of bilinear complexity for batch matrix multiplication to further reduce the number of workers by a factor of 2. However, here we focus on demonstrating the coding gain and present the results for general scenarios.

are linear coding schemes that achieve a recovery threshold of $2LR(p,m,n)-1$. *Moreover, for extended settings in batch matrix multiplication, linear coding schemes achieve the following recovery thresholds:*

- *One sided* T*-security:* $2LR(p,m,n)+T-1$,
- *Fully* T*-security:* $2LR(p,m,n)+2T-1$,
- *Privacy of request:* $2LR(p,m,n)$,
- *Security and Privacy:* $2LR(p,m,n)+1$,
- *Full Privacy:* $2LR(p,m,n)+1$.

Remark 9. Entangled polynomial codes provide coding schemes that order-wise improves the state-of-the-art schemes in [20], [21] for batch matrix multiplication when the matrices are block-wise partitioned. The coding designs proposed in [20], [21] focused on straggler mitigation and requires a recovery threshold of at least $Lpmn$, with computation and storage costs equal or greater than the framework considered in this paper; while entangled polynomial codes achieve order-wise smaller recovery thresholds for any large p, m and n.

Remark 10. The main proof idea is to note that batch-multiplying L pairs of matrices is still computing a bilinear function, so one can simply use similar bilinear decomposition bounds for this operation as in [42], and all earlier achievability and converse results extend to batch computation. However, to better demonstrate the achievability of subcubic recovery thresholds, we present our results based on a subadditivity upper bound. Specifically, let $R(L,p,m,n)$ denote the bilinear complexity of batch multiplying L pairs of m-by-p and p-by-n matrices, it satisfies $R(L,p,m,n)\leq LR(p,m,n)$. More generally, one can obtain achievability and converse results in batch matrix multiplication by simply substituting the quantity $R(p,m,n)$ in results for single matrix multiplication with $R(L,p,m,n)$. One can similarly prove the factor-of-2 optimalities for the general entangled polynomial codes framework for all settings we presented for batch matrix multiplication.

V. Achievability Schemes for Secure Distributed Matrix Multiplication

In this section, we present coding schemes for the simple scenario where the only additional requirement for distributed matrix multiplication is to maintain the security of input matrices. This provides a proof for Theorem 1.

Given parameters p, m, and n, we denote the partitioned uncoded input matrices by $\{A_{i,j}\}_{i\in[p],j\in[m]}$ and $\{B_{i,j}\}_{i\in[p],j\in[n]}$. The encoding consists of two steps.

In Step 1, given any upper bound construction of $R(p,m,n)$ (e.g., Strassen's construction) with rank R and tensor tuples $a\in\mathbb{F}^{R\times p\times m}$, $b\in\mathbb{F}^{R\times p\times n}$, and $c\in\mathbb{F}^{R\times m\times n}$, we pre-encode the inputs each into a list of R coded submatrices.[14]

$$\tilde{A}_{i,\text{vec}}\triangleq\sum_{j,k}A_{j,k}a_{ijk},\qquad \tilde{B}_{i,\text{vec}}\triangleq\sum_{j,k}B_{j,k}b_{ijk}. \tag{14}$$

As we have explained in [42], this pre-encoding essentially provides a linear coding scheme with R workers that does not provide straggler-resiliency and data-security, of which we need to take into account in the second part of the encoding.

In Step 2, note that it suffices to recover the element-wise products $\tilde{A}_{1,\text{vec}}^{\intercal}\tilde{B}_{1,\text{vec}},...,\tilde{A}_{R,\text{vec}}^{\intercal}\tilde{B}_{R,\text{vec}}$. We can build upon optimal coding constructions for element-wise multiplication, first presented in [15] for straggler mitigation and then extended in [2] to also provide data-privacy.

We first pad the two vectors $\{\tilde{A}_{i,\text{vec}}\}_{i\in[R]}$ and $\{\tilde{B}_{i,\text{vec}}\}_{i\in[R]}$ with uniformly random keys. If matrix A needs to be stored securely against up to T colluding workers, we pad the pre-coded matrices of A with T uniformly random matrices $Z_1,...,Z_T\in\mathbb{F}^{\frac{s}{p}\times\frac{t}{m}}$. Explicitly, we define

$$\tilde{\boldsymbol{A}}'_{\text{vec}}\triangleq(\tilde{A}_{1,\text{vec}},...,\tilde{A}_{R,\text{vec}},Z_1,...,Z_T) \tag{15}$$

if A needs to be stored securely; otherwise, we define

$$\tilde{\boldsymbol{A}}'_{\text{vec}}\triangleq(\tilde{A}_{1,\text{vec}},...,\tilde{A}_{R,\text{vec}}). \tag{16}$$

Similarly, we define vector $\tilde{\boldsymbol{B}}'_{\text{vec}}$ for matrix B in the same way. For brevity, we denote the lengths of $\tilde{\boldsymbol{A}}'_{\text{vec}}$ and $\tilde{\boldsymbol{B}}'_{\text{vec}}$ by L_A and L_B.

Then we arbitrarily select $R+T$ distinct elements from $\mathbb{F}$, denoted $x_1,...,x_{R+T}$, and N distinct elements from $\mathbb{F}\backslash\{x_1,...,x_R\}$, denoted $y_1,...,y_N$. We encode the inputs for each worker i as follows.

$$\tilde{A}_i=\sum_{j\in[L_A]}\tilde{A}'_{j,\text{vec}}\cdot\prod_{k\in[L_A]\backslash j}\frac{(y_i-x_k)}{(x_j-x_k)}, \tag{17}$$

$$\tilde{B}_i=\sum_{j\in[L_B]}\tilde{B}'_{j,\text{vec}}\cdot\prod_{k\in[L_B]\backslash j}\frac{(y_i-x_k)}{(x_j-x_k)}. \tag{18}$$

[14]For detailed definitions of bilinear complexity and upper bound constructions, see [42].

As proved in [2], the above encoding scheme satisfies the requirements for both one-sided and fully T-secure settings.[15] [16]

According to the PCC framework, we have encoded the input matrices using polynomials with degrees of L_A-1 and L_B-1, where each worker i is assigned their evaluations at y_i. Hence, after the workers multiply their coded matrices, they obtain evaluations of the multiplicative product of these polynomials, which has degree L_A+L_B-2. Note that evaluations of this composed polynomial at $x_1,...,x_R$ recovers the needed element-wise products. The decodability requirement of PCC is satisfied. Consequently, the master can recover the final output by interpolating the composed polynomial after sufficiently many results from the workers are received, achieving a recovery threshold of L_A+L_B-1.

Recall that for one-sided T-secure setting, we have $L_A=R+T$ and $L_B=R$; then for fully T-secure setting, we have $L_A=L_B=R+T$. Hence, we have obtained linear coding schemes with recovery thresholds of $2R+T-1$ and $2R+2T-1$ for both settings respectively given any upper bound constructions of $R(p,m,n)$ with rank R. Fundamentally, there exist constructions that exactly achieve the rank $R(p,m,n)$, which proves the existence of coding schemes stated in Theorem 1.

Remark 11. The coding scheme we presented for computing element-wise product with one-sided privacy naturally extends to provide optimal codes for the scenario of batch computation of multilinear functions where each of the input entries are coded to satisfy possibly different security requirements.

VI. Achievability Schemes for Private Distributed Matrix Multiplication

In this section, we present the coding scheme for proving Theorem 2 and 3. We start with the setting for Theorem 2, where the goal is to multiply matrix A by one of the matrices from $B^{(1)},...,B^{(M)}$.

Similar to Section V, we first pre-encode the input matrices into lists of vectors of length R, given any upper bound construction of $R(p,m,n)$ with rank R and tensor tuples $a\in\mathbb{F}^{R\times p\times m}$, $b\in\mathbb{F}^{R\times p\times n}$, and $c\in\mathbb{F}^{R\times m\times n}$. In particular, given parameters p, m, and n, we denote the partitioned uncoded input matrices by $\{A_{i,j}\}_{i\in[p],j\in[m]}$ and $\{B^{(\ell)}_{i,j}\}_{i\in[p],j\in[n],\ell\in[M]}$. We define

$$\tilde{A}_{i,\text{vec}}\triangleq\sum_{j,k}A_{j,k}a_{ijk},\qquad \tilde{B}^{(\ell)}_{i,\text{vec}}\triangleq\sum_{j,k}B^{(\ell)}_{j,k}b_{ijk},\tag{19}$$

for each $i\in[R]$ and $\ell\in[M]$. Then given any request $D\in[M]$, it suffices to compute the element-wise product $\{\tilde{A}^{\intercal}_{i,\text{vec}}\tilde{B}^{(D)}_{i,\text{vec}}\}_{i\in[R]}$ while keeping D private.

For the second part of the encoding scheme, we present a new coding construction for computing element-wise product with privacy, which is motivated by ideas developed in [13], [15] and earlier sections. In particular, we first pad the pre-encoded vector of A with random keys for security. We define

$$\tilde{\boldsymbol{A}}'_{\text{vec}}\triangleq(\tilde{A}_{1,\text{vec}},...,\tilde{A}_{R,\text{vec}},Z),\tag{20}$$

if A needs to be stored securely, where Z is a random key sampled from $\mathbb{F}^{\frac{s}{p}\times\frac{t}{m}}$ with a uniform distribution; otherwise,

$$\tilde{\boldsymbol{A}}'_{\text{vec}}\triangleq(\tilde{A}_{1,\text{vec}},...,\tilde{A}_{R,\text{vec}}).\tag{21}$$

For brevity, we denote the length of $\tilde{\boldsymbol{A}}'_{\text{vec}}$ by L_A.

We arbitrarily select $R+1$ distinct elements from $\mathbb{F}$, denoted $x_1,...,x_{R+1}$, and encode matrix A by defining the following Lagrange polynomial,

$$\tilde{A}(x)\triangleq\sum_{i\in[L_A]}\tilde{A}'_{i,\text{vec}}\cdot\prod_{j\in[L_A]\backslash i}\frac{(x-x_j)}{(x_i-x_j)},\tag{22}$$

We then arbitrarily select a finite subset $\mathcal{Y}$ of $\mathbb{F}\backslash\{x_1,...,x_R\}$ with at least N elements, and let the master uniformly randomly generate N distinct elements from $\mathcal{Y}$, denoted $y_1,...,y_N$. The master sends $\tilde{A}_i=\tilde{A}(y_i)$ to each worker i, which satisfies the security of A when required.

Given a request D, we similarly define

$$\tilde{B}(x)\triangleq\sum_{i\in[R+1]}\tilde{B}'_{i,\text{vec}}\cdot\prod_{j\in[R+1]\backslash i}\frac{(x-x_j)}{(x_i-x_j)},\tag{23}$$

[15] Such property is referred to as T-private in [2], [58].

[16] As a simple proof, note that when security is required for any input, say matrix A, the coded variables sent to any subset of T workers are padded by some linear combinations of T uniformly random keys, which are jointly uniformly random. In particular, the padded variable for each worker i equals a degree-$(T-1)$ Lagrange interpolated version of random keys evaluated at distinct points and multiplied by non-zero constant factors $\prod_{k\in[R]}\frac{(y_i-x_k)}{(x_j-x_k)}$. Using Lagrange interpolation, any set of T padded variables recovers all T random keys, hence must also be uniformly random, which provides the needed security.

where

$$\tilde{\boldsymbol{B}}'_{\text{vec}} \triangleq (\tilde{B}^{(D)}_{1,\text{vec}}, ..., \tilde{B}^{(D)}_{R,\text{vec}}, Y), \tag{24}$$

and $Y \in \mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$ is a quantity to be specified later. If the encoding can be designed such that each worker essentially computes $\tilde{A}^{\intercal}(y_i)\tilde{B}(y_i)$, then we can achieve the recovery thresholds stated in Theorem 2.

To construct a private computing scheme where $\tilde{B}_i$ is equivalent to $\tilde{B}(y_i)$, we divide[17] $\tilde{B}(x)$ by a scalar[18]

$$c(x) \triangleq \prod_{j\in[R]} \frac{(x-x_j)}{(x_{R+1}-x_j)}, \tag{25}$$

so that the result can be expressed as the unweighted sum of Y and $\tilde{B}^{(D)}_{\text{Norm}}(x)$ with function $\tilde{B}^{(\cdot)}_{\text{Norm}}(x)$ defined as follows

$$\tilde{B}^{(k)}_{\text{Norm}}(x) \triangleq -\sum_{i\in[R]} \tilde{B}^{(k)}_{i,\text{vec}} \left(\prod_{j\in[R]\setminus i} \frac{(x_{R+1}-x_j)}{(x_i-x_j)}\right) \frac{(x-x_{R+1})}{(x-x_i)}. \tag{26}$$

We let the master generate i.i.d. uniformly random variables $\{z_i\}_{i\in[M]\setminus D}$ from $\mathcal{Y}$ independent of y_i's. The master sends a query $Q_i = (q_{i1}, ..., q_{iM})$ to each worker i with $q_{ij} = y_i$ for $j = D$ and $q_{ij} = z_j$ for $j \neq D$. Because each Q_i appears uniformly random to worker i, the presented coding scheme satisfies the privacy requirement.

We let each worker i encode $\boldsymbol{B}$ by computing $\sum_j \tilde{B}^{(j)}_{\text{Norm}}(q_{ij})$. Consequently, each encoded variable can be re-expressed as

$$\tilde{B}_i = \frac{\tilde{B}(y_i)}{c(y_i)} \tag{27}$$

with $Y = \sum_{j\neq D} \tilde{B}^{(j)}_{\text{Norm}}(q_{ij})$ independent of y_i.

After the workers multiply the coded matrices, each worker i essentially returns $\tilde{A}^{\intercal}(y_i)\tilde{B}(y_i)/c(y_i)$. Because y_i is available at the decoder, the master can decode $\tilde{A}^{\intercal}(y_i)\tilde{B}(y_i)$ given each worker i's returned result by computing $c(y_i)$. Hence, by receiving results from sufficiently many workers, the master can recover the needed element-wise product by Lagrange interpolating the polynomial $\tilde{A}^{\intercal}(x)\tilde{B}(x)$, and proceed to compute the final output.

Because the degree of $\tilde{A}^{\intercal}(x)\tilde{B}(x)$ equals $L_A - 1 + R$, the presented coding scheme achieves a recovery threshold of $L_A + R$. Note that $L_A = R$ when no security is required and $L_A = R + 1$ when A is stored securely. We have obtained linear coding schemes with recovery thresholds of $2R-1$ for private coded matrix multiplication, and $2R$ for private and secure distributed matrix multiplication for any upper bound construction of $R(p, m, n)$, which completes the proof for Theorem 2.

Remark 12. This coding scheme naturally extends to the scenario where the encoding of A is required to be T-secure. A recovery threshold of $2R(p, m, n) + T$ can be achieved, which is optimal within a factor of 2.[19]

We now present the coding scheme for the fully private setting. The matrices are pre-encoded the same way and we denote the corresponding matrices by $\{\tilde{A}^{(\ell)}_{i,\text{vec}}, \tilde{B}^{(\ell)}_{i,\text{vec}}\}_{i\in[R],\ell\in[M]}$. To recover the final output, it suffices to compute $\{\tilde{A}^{(D)\intercal}_{i,\text{vec}}\tilde{B}^{(D)}_{i,\text{vec}}\}_{i\in[R]}$.

We arbitrarily select $R+1$ distinct elements from $\mathbb{F}$, denoted $x_1, ..., x_{R+1}$, and define the following functions

$$\tilde{A}^{(k)}_{\text{Norm}}(x) \triangleq -\sum_{i\in[R]} \tilde{A}^{(k)}_{i,\text{vec}} \left(\prod_{j\in[R]\setminus i} \frac{(x_{R+1}-x_j)}{(x_i-x_j)}\right) \frac{(x-x_{R+1})}{(x-x_i)}.$$

$$\tilde{B}^{(k)}_{\text{Norm}}(x) \triangleq -\sum_{i\in[R]} \tilde{B}^{(k)}_{i,\text{vec}} \left(\prod_{j\in[R]\setminus i} \frac{(x_{R+1}-x_j)}{(x_i-x_j)}\right) \frac{(x-x_{R+1})}{(x-x_i)}.$$

We then arbitrarily select a finite subset $\mathcal{Y}$ of $\mathbb{F}\setminus\{x_1, ..., x_R\}$ with at least N elements. Let the master uniformly randomly generate N distinct elements from $\mathcal{Y}$, denoted $y_1, ..., y_N$, and i.i.d. uniformly random variables $\{z_i\}_{i\in[M]\setminus D}$ from $\mathcal{Y}$ independent of y_i's. The master sends a query $Q_i = (q_{i1}, ..., q_{iM})$ to each worker i with $q_{ij} = y_i$ for $j = D$, and $q_{ij} = z_j$ for $j \neq D$. This query is

[17]Throughout the proof, the divisor $c(x)$ will only be used for inputs $x \in \mathcal{Y}$, which is disjoint with $\{x_1, ..., x_R\}$. Hence, it is always non-zero.

[18]Note that here we are exploiting the fact that each worker computes a function that is multilinear. For more general scenarios (e.g., general polynomial evaluations we considered in [2]), scaling the coded variables could affect decodability.

[19]In particular, any linear code requires at least $R(p, m, n) + T$ workers as proved in Appendix A.

fully private, because for each worker i, $q_{i1}, ..., q_{iM}$ appears i.i.d. uniformly random in $\mathcal{Y}$.

Each worker i encodes the input matrices as follows

$$\tilde{A}_i = \sum_j \tilde{A}^{(j)}_{\text{Norm}}(q_{ij}), \tag{28}$$

$$\tilde{B}_i = \sum_j \tilde{B}^{(j)}_{\text{Norm}}(q_{ij}). \tag{29}$$

After the computation result is received from any worker i, by multiplying a scalar factor $c^2(y_i)$ with function c defined in equation (25), the master recovers the evaluation of the product of two Lagrange polynomials of degree R at point y_i. By interpolating this polynomial and re-evaluating it at x_i's, the master can recover all needed element-wise products. This provides a coding scheme that proves Theorem 3.

VII. Achievability Schemes for Batch Distributed Matrix Multiplication

In this section, we present the coding scheme for proving Theorem 4. We start with the basic setting where no security or privacy is required. As mentioned in Section IV-C, one can directly decompose the tensor characterizing the L-batch matrix multiplication, and all earlier results as well as Theorem 3 in [15] extend to batch distributed matrix multiplication. However, we instead present one certain class of upper bounds based on the subadditivity of tensor rank.

Explicitly, we denote the partitioned uncoded input matrices by $\{A^{(k)}_{i,j}\}_{i\in[p],j\in[m],k\in[L]}$ and $\{B^{(k)}_{i,j}\}_{i\in[p],j\in[n],k\in[L]}$. Given any upper bound construction of $R(p,m,n)$ with rank R and tensor tuples $a \in \mathbb{F}^{R\times p\times m}$, $b \in \mathbb{F}^{R\times p\times n}$, and $c \in \mathbb{F}^{R\times m\times n}$, we define

$$\tilde{A}_{i,\ell,\text{vec}} \triangleq \sum_{j,k} A^{(\ell)}_{j,k} a_{ijk}, \qquad \tilde{B}_{i,\ell,\text{vec}} \triangleq \sum_{j,k} B^{(\ell)}_{j,k} b_{ijk}. \tag{30}$$

for each $i \in [R]$ and $\ell \in [L]$. Note that the batch product can be recovered from the element-wise product $\{\tilde{A}^{\intercal}_{i,\ell,\text{vec}}\tilde{B}_{i,\ell,\text{vec}}\}_{i\in[R],\ell\in[L]}$. One can directly apply the optimal coding scheme presented in [15], which encodes the pre-encoded vectors using Lagrange polynomials. According to Corollary 1 in [15], the resulting scheme achieves a recovery threshold of $2LR-1$, which proves the basic scenario for Theorem 4.

Remark 13. In [59], Lagrange encoding is also applied to compute inner product (sum of element-wise products) to achieve the same recovery threshold. Remarkably, [59] pointed out that the encoding can be made systematic as Lagrange polynomials pass through all uncoded inputs, as stated in [60]. It is mentioned in [59] that the main benefit of using systematic encoding designs is to enable recovery from results of a certain smaller subset of "systematic" workers, which provides backward-compatibility and potentially reduces computation and decoding latency. Based on this observation, the entangled polynomial codes can be adapted to a "systematic" version that goes beyond inner product and handles generalized block-wise partitioned matrices by choosing the same evaluation points as in [60], so that a subset of $R(p,m,n)$ workers computes all needed "uncoded" products of the pre-encoded matrices, and all major benefits of systematic encoding are provided. This construction gives a practical solution to an open problem stated in [61], in the sense of achieving all major benefits of systematic encoding, and improving recovery thresholds for any sufficiently large values of p, m, and n.

Now we formally state the settings with security and privacy requirements. Similar to Section IV, for batch matrix multiplication with security requirement, the formulation is the same as the basic setup for batch distributed matrix multiplication, except that the inputs need to be stored information-theoretic privately even if up to T workers collude.

Definition 6. *For batch distributed matrix multiplication, a coding scheme is one-sided T-secure, if*

$$I(\{\tilde{A}_i, \tilde{B}_i\}_{i\in\mathcal{T}}; \boldsymbol{A}) = 0 \tag{31}$$

for any subset $\mathcal{T}$ with a size of at most T, where A is generated uniformly at random.

Definition 7. *For batch distributed matrix multiplication, a coding scheme is fully T-secure, if*

$$I(\{\tilde{A}_i, \tilde{B}_i\}_{i\in\mathcal{T}}; \boldsymbol{A}, \boldsymbol{B}) = 0 \tag{32}$$

is satisfied for any $|\mathcal{T}| \le T$, for uniformly randomly generated A and B.

When privacy is taken into account, the goal is to instead batch multiply a list of L matrices by one

unknown subset of L matrices $\boldsymbol{B} = \{B^{(i,D)}\}_{i\in[L]}$ from a set $\boldsymbol{B} = \{B^{(i,j)}\}_{i\in[L],j\in[M]}$ while keeping the request D private to the workers. The master sends a query and a coded version of $\boldsymbol{A}$ with size $\mathbb{F}^{\frac{s}{p}\times\frac{t}{m}}$ to each worker, then each worker encodes matrices $\boldsymbol{B}$ into a coded submatrix of size $\mathbb{F}^{\frac{s}{p}\times\frac{r}{n}}$ based on the query, the same as in private distributed matrix multiplication.

Definition 8. *For batch matrix multiplication, a computing scheme is private, if*

$$I(D; Q_i, \tilde{A}_i, \boldsymbol{B}) = 0 \tag{33}$$

for any $i \in [N]$, where $\boldsymbol{A}, \boldsymbol{B}, D$ are sampled uniformly at random.

Furthermore,

Definition 9. *A computing scheme for batch matrix multiplication is private and secure, if it is private, and also satisfies,*

$$I(\tilde{A}_i; \boldsymbol{A}) = 0 \tag{34}$$

for any $i \in [N]$ when $\boldsymbol{A}$ is sampled uniformly random.

Finally, for fully private batch matrix multiplication, the goal is to batch multiply L pairs of matrices given two lists of inputs $\boldsymbol{A} = \{A^{(i,j)}\}_{i\in[L],j\in[M]}$ and $\boldsymbol{B} = \{B^{(i,j)}\}_{i\in[L],j\in[M]}$. The master aims to compute $\{A^{(i,D)\intercal}B^{(i,D)}\}_{i\in[L]}$ given an index D, while keeping D private. The rest of the computation follows similarly to the fully private and the batch distributed matrix multiplication frameworks. Explicitly, let Q_i denote the query the master sends to worker i, we have the following requirement.

Definition 10. *A computing scheme is fully private, if*

$$I(D; Q_i, \boldsymbol{A}, \boldsymbol{B}) = 0 \tag{35}$$

for any $i \in [N]$, where $\boldsymbol{A}, \boldsymbol{B}, D$ are sampled uniformly at random.

The achievability schemes for all these settings can be built based on coding ideas we presented earlier in this paper. In particular, by first pre-encoding each of the input matrices using any upper bound construction of $R(p, m, n)$, the task of batch-multiplying L matrices is reduced to computing element-wise product of two vectors of lengths at most $LR(p, m, n)$. Then observe that in the second parts of all coding schemes we presented in earlier sections for non-batch matrix multiplication, we essentially provided linear codes that compute element-wise products of vectors of any sizes. By directly applying those proposed designs to the extended pre-coded vectors for batch multiplication, we obtain the needed computing schemes for proving Theorem 4 where the achieved recovery threshold upper bounds are stated by swapping $R(p, m, n)$ into $LR(p, m, m)$.

VIII. Conclusion and Future Directions

In this paper, we provided an overview of coded computation, and showed that entangled polynomial codes as an effective approach for computing block matrix multiplication can be applied beyond straggler mitigation. We investigated three important settings: secure, private, and batch distributed matrix multiplication, and demonstrated the effectiveness of entangled polynomial codes in providing unified solutions with order-wise improvements upon the state of the arts. To demonstrate the coding gain, we focus on generalizing the second version of the entangled polynomial code, the one that achieves a recovery threshold of $2R(p, m, n)-1$. Note that similar to straggler mitigation, where a "cubic" recovery thresholds $pmn + p - 1$ can be achieved when any of p, m, n is small, one should expect that similar optimal coding designs can also be developed for the settings we studied in this work, and it is an interesting follow-up direction to establish those constructions.

Another research direction is to consider general computation tasks beyond matrix multiplication. Entangled polynomial codes enables exploiting the idea that any bilinear function can be characterized by a rank-3 tensor, and any decomposition of the tensor reduces the bilinear function into batch evaluations of simpler computation tasks that can be assigned to distributed workers and effectively computed using Lagrange encoding. This approach directly generalizes to any multilinear functions. However, it remains as future work to design optimal codes for general computation even for linear coding functions.

Finally, an interesting direction is to find practical non-linear codes for general computing scenarios. On one hand, linear codes guarantee that the encoding and decoding complexities grow linearly with respect to the input and output size, which ensures

negligible coding overhead when the dataset is large and a significant amount of computation is assigned to each worker. On the other hand, non-linear codes are known to allow achieving the same recoverability results as classical erasure/secret-sharing codes (e.g., by having workers directly forwarding the coded data to the master), but could introduce a significant computational delay. It is an open research direction to find non-linear codes in general with encoding and decoding functions satisfying complexity constraints, while improving linear codes in recovery threshold. Some recent applications of coded computing has been developed on privacy-preserving machine learning [7] and verifiable computing [62], [63].

ACKNOWLEDGEMENTS

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, ONR Award No. N00014-16-1-2189, and CCF-1763673. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Qian Yu is supported by the Google PhD Fellowship.

APPENDIX A
OPTIMALITY PROOFS FOR ENTANGLED POLYNOMIAL CODES

In this appendix, we prove converse lemmas that are needed for optimality results stated in Remark 5, 7, 8, and 12. Particularly, Remark 5, 7, and 12 can be proved based on the following lemma.

Lemma 1. *For coded distributed matrix multiplication with parameters p, m, and n, if the encoding of any input matrix needs to T-secure, any linear code requires using at least $R(p,m,n)+T$ workers.*

And Remark 8 follows from the following lemma.

Lemma 2. *For fully private coded matrix multiplication with parameters p, m, and n, any linear code requires using at least $R(p,m,n)+1$ workers.*

Formally, any computing scheme is linear if each (or each list of) input matrix is encoded by computing linear combinations of their submatrices and a possible list of uniformly random keys, and the decoding functions compute linear combinations of workers outputs. When the privacy of request is required, the coefficients of the linear combinations could depend on the query $Q_i's$ sent by the master. We present the proofs of the above lemmas as follows.

Proof of Lemma 1: Without loss of generality, we assume A needs to be encoded T-securely. Consider any valid computing scheme. If there are privacy requirements, we focus on the computing scenario where $D=1$ and the queries are any fixed possible values. In this case, we are multiplying A by a single know matrix, and we denote it by B for all possible settings; moreover, the coefficients in encoding and decoding functions are fixed. If B also needs to be stored securely, we focus on the computing scenario where the random keys used for encoding B are all zero. Under these conditions, B is encoded as if no random keys are used.

Recall that from the T-security requirement, the collection of coded variables for matrix A assigned to any subset of at most T workers needs to be independent of A. By the assumption of linear codes, for any such given subset, we can focus on values of random keys that are fixed linear combinations of submatrices of A, so that the values of these coded variables are all zero while A can be arbitrary. As a consequence, the workers within this subset return constant 0 and the master essentially recovers the final output only based on the rest of available results.

Because the final output is variable, the considered computing scheme requires more than T workers. By choosing a subset of size T, we have a set of linear decoding function that recovers the final output from $N-T$ workers, where both A and B is coded linearly and deterministically. From the definition of bilinear complexity, the matrix product $A^{\intercal}B$ is recoverable from products of linearly and deterministically coded variables only if the number of products is at least $R(p,m,n)$. Combining these facts, we have $N \geq R(p,m,n)+T$. ■

Proof of Lemma 2: Recall that we assumed the non-trivial cases where $M>1$. Given any valid computing scheme, we can focus on the scenario where $D=1$ or 2. To allow decodability of $A^{(2)\intercal}B^{(2)}$, there must be a worker i and a possible value of query Q_i for $D=2$ such that the coded variable $\tilde{A}_i$ computes a linear combination with

a non-zero component from $A^{(2)}$. Because Q_i is independent of D due to privacy requirement, we can consider a possible fixed instance of the queries for $D = 1$ with Q_i taking the same value, resulting in the same encoding functions for worker i.

Now by setting the values of submatrices of $A^{(2)}, ..., A^{(M)}$ be some fixed linear combination of submatrices of $A^{(1)}$, worker i returns constant 0 and the master essentially decodes based on results from other $N - 1$ workers. Furthermore, let $B^{(2)}, ..., B^{(M)} = 0$, the coded variables for the result $N - 1$ workers become deterministically and linearly coded version of $A^{(1)}$ and $B^{(1)}$. As we have mentioned in proof of Lemma 1, this indicates $N - 1 \geq R(p, m, n)$. Thus, $N \geq R(p, m, n) + 1$. ■

REFERENCES

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proceedings of Machine Learning Research* (K. Chaudhuri and M. Sugiyama, eds.), vol. 89 of *Proceedings of Machine Learning Research*, pp. 1215–1225, PMLR, 16–18 Apr 2019, *arXiv:1806.00939*, 2018.

[3] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, p. 222–238, Aug. 1983.

[4] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, p. 382–401, July 1982.

[5] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and privacy in cloud computing: A survey," in *2010 Sixth International Conference on Semantics, Knowledge and Grids*, pp. 105–112, Nov 2010.

[6] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 19–38, May 2017.

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

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *e-print arXiv:1512.02673*, 2015.

[9] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, pp. 2092–2100, 2016.

[10] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, pp. 4406–4416, Curran Associates, Inc., 2017, *arXiv:1705.10464*, 2017.

[11] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," *arXiv preprint arXiv:1806.00469*, 2018.

[12] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 141–150, Jan 2019.

[13] M. Kim and J. Lee, "Private secure coded computation," *arXiv preprint arXiv:1902.00167*, 2019.

[14] W.-T. Chang and R. Tandon, "On the upload versus download cost for secure and private matrix multiplication," *arXiv preprint arXiv:1906.10684*, 2019.

[15] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrxix multiplication: Fundamental limits and optimal coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 2022–2026, June 2018, *arXiv:1801.07487v1*, 2018.

[16] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1264–1270, Oct 2017.

[17] M. Aliasgari, O. Simeone, and J. Kliewer, "Distributed and private coded matrix computation with flexible communication load," *arXiv preprint arXiv:1901.07705*, 2019.

[18] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *arXiv preprint arXiv:1909.00407*, 2019.

[19] H. A. Nodehi, S. R. H. Najarkolaei, and M. A. Maddah-Ali, "Entangled polynomial coding in limited-sharing multi-party computation," in *2018 IEEE Information Theory Workshop (ITW)*, pp. 1–5, Nov 2018.

[20] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch matrix multiplication," *arXiv preprint arXiv:1909.13873*, 2019.

[21] Z. Jia and S. A. Jafar, "Generalized cross subspace alignment codes for coded distributed batch matrix multiplication," 2019.

[22] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, Aug 1969.

[23] V. Y. Pan, "Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations," in *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pp. 166–176, Oct 1978.

[24] J. Hopcroft and L. Kerr, "On minimizing the number of multiplications necessary for matrix multiplication," *SIAM Journal on Applied Mathematics*, vol. 20, no. 1, pp. 30–36, 1971.

[25] J. D. Laderman, "A noncommutative algorithm for multiplying $3\times$ 3 matrices using 23 multiplications," *Bulletin of the American Mathematical Society*, vol. 82, no. 1, pp. 126–128, 1976.

[26] S. Winograd, "On multiplication of 2×2 matrices," *Linear Algebra and its Applications*, vol. 4, no. 4, pp. 381 – 388, 1971.

[27] D. Bini, "Relations between exact and approximate bilinear algorithms. applications," *CALCOLO*, vol. 17, pp. 87–97, Jan 1980.

[28] A. Schönhage, "Partial and total matrix multiplication," *SIAM Journal on Computing*, vol. 10, pp. 434–455, aug 1981.

[29] F. Romani, "Some properties of disjoint sums of tensors related to matrix multiplication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 263–267, 1982.

[30] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," in *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*, SFCS '81, (Washington, DC, USA), pp. 82–90, IEEE Computer Society, 1981.

[31] V. Strassen, "The asymptotic spectrum of tensors and the exponent of matrix multiplication," in *Proceedings of the 27th*

Annual Symposium on Foundations of Computer Science, SFCS '86, (Washington, DC, USA), pp. 49–54, IEEE Computer Society, 1986.

[32] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251 – 280, 1990. Computational algebraic complexity editorial.

[33] J. Landsberg, "The border rank of the multiplication of 2× 2 matrices is seven," *Journal of the American Mathematical Society*, vol. 19, no. 2, pp. 447–459, 2006.

[34] A. J. Stothers, *On the complexity of matrix multiplication*. PhD thesis, University of Edinburgh, 2010.

[35] C.-E. Drevet, M. Nazrul Islam, and E. Schost, "Optimization techniques for small matrix multiplication," *Theoretical Computer Science*, vol. 412, no. 22, pp. 2219–2236, 2011.

[36] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *In Proc. 44th ACM Symposium on Theory of Computation*, pp. 887–898, 2012.

[37] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic complexity theory*, vol. 315. Springer Science & Business Media, 2013.

[38] A. V. Smirnov, "The bilinear complexity and practical algorithms for matrix multiplication," *Computational Mathematics and Mathematical Physics*, vol. 53, pp. 1781–1795, Dec 2013.

[39] A. Sedoglavic, "A non-commutative algorithm for multiplying 5×5 matrices using 99 multiplications," *arXiv preprint arXiv:1707.06860*, 2017.

[40] A. Sedoglavic, "A non-commutative algorithm for multiplying (7 × 7) matrices using 250 multiplications," *arXiv preprint arXiv:1712.07935*, 2017.

[41] F. Le Gall, "Complexity of matrix multiplication and bilinear problems," [Online] https://conferences.mpi-inf.mpg.de/adfocs-17/material/FLG_L1.pdf.

[42] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Transactions on Information Theory*, vol. 66, pp. 1920–1933, March 2020.

[43] J. Kakar, S. Ebadifar, and A. Sezgin, "Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation," *arXiv preprint arXiv:1810.13006*, 2018.

[44] R. G. L. D'Oliveira, S. El Rouayheb, and D. Karpuk, "Gasp codes for secure distributed matrix multiplication," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1107–1111, July 2019.

[45] J. Kakar, S. Ebadifar, and A. Sezgin, "On the capacity and straggler-robustness of distributed secure matrix multiplication," *IEEE Access*, vol. 7, pp. 45783–45799, 2019.

[46] S. Ebadifar, J. Kakar, and A. Sezgin, "The need for alignment in rate-efficient distributed two-sided secure matrix computation," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2019.

[47] H. A. Nodehi and M. A. Maddah-Ali, "Secure coded multi-party computation for massive matrix operations," *arXiv preprint arXiv:1908.04255*, 2019.

[48] Z. Jia and S. A. Jafar, "On the capacity of secure distributed matrix multiplication," *arXiv preprint arXiv:1908.06957*, 2019.

[49] J. Kakar, A. Khristoforov, S. Ebadifar, and A. Sezgin, "Uplink-downlink tradeoff in secure distributed matrix multiplication," *arXiv preprint arXiv:1910.13849*, 2019.

[50] R. G. D'Oliveira, S. El Rouayheb, D. Heinlein, and D. Karpuk, "Degree tables for secure distributed matrix multiplication," 2019.

[51] M. Kim, H. Yang, and J. Lee, "Private coded matrix multiplication," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2019.

[52] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2403–2407, June 2017.

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

[54] H. Jeong, T. M. Low, and P. Grover, "Coded fft and its communication overhead," *arXiv preprint arXiv:1805.09891*, 2018.

[55] J. Von Zur Gathen and J. Gerhard, *Modern computer algebra*. Cambridge university press, 2013.

[56] M. Bläser, *Fast Matrix Multiplication*. No. 5 in Graduate Surveys, Theory of Computing Library, 2013.

[57] Q. Yu and A. S. Avestimehr, "Harmonic coding: An optimal linear code for privacy-preserving gradient-type computation," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1102–1106, July 2019.

[58] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, (New York, NY, USA), p. 1–10, Association for Computing Machinery, 1988.

[59] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, pp. 278–301, Jan 2020, *arXiv:1801.10292*, 2018.

[60] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, pp. 4661–4676, Aug 2014.

[61] H. Jeong, Y. Yang, and P. Grover, "Systematic matrix multiplication codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1–5, July 2019.

[62] S. Sahraei and A. S. Avestimehr, "Interpol: Information theoretically verifiable polynomial evaluation," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1112–1116, 2019.

[63] S. Sahraei and S. Avestimehr, "Infocommit: Information-theoretic polynomial commitment and verification," *arXiv preprint arXiv:2002.00559*, 2020.

PLACE PHOTO HERE

Qian Yu Qian Yu received his Ph.D. degree in Electrical and Computer Engineering at University of Southern California (USC), Viterbi School of Engineering. He received an M.Eng. degree in Electrical Engineering and a B.S. degree in EECS and Physics, both from Massachusetts Institute of Technology (MIT). His interests span information theory, distributed computing, and many other problems math-related.

Qian is a recipient of the Google PhD Fellowship in 2018, and received the Jack Keil Wolf ISIT Student Paper Award in 2017. He was an MHI PhD scholar at USC.

PLACE PHOTO HERE

Salman Avestimehr A. Salman Avestimehr is a Professor and director of the Information Theory and Machine Learning (vITAL) research lab at the Electrical and Computer Engineering Department of University of Southern California. He received his Ph.D. in 2008 and M.S. degree in 2005 in Electrical Engineering and Computer Science, both from the University of California, Berkeley. Prior to that, he obtained his B.S. in Electrical Engineering from Sharif University of Technology in 2003. His research interests include information theory and coding theory, and large-scale distributed computing and machine learning, secure and private computing, and blockchain systems.

Dr. Avestimehr has received a number of awards for his research, including the James L. Massey Research & Teaching Award from IEEE Information Theory Society, an Information Theory Society and Communication Society Joint Paper Award, a Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House (President Obama), a Young Investigator Program (YIP) award from the U. S. Air Force Office of Scientific Research, a National Science Foundation CAREER award, the David J. Sakrison Memorial Prize, and several Best Paper Awards at Conferences. He has been an Associate Editor for IEEE Transactions on Information Theory and a general Co-Chair of the 2020 International Symposium on Information Theory (ISIT). He is a fellow of IEEE.