Secure Distributed Matrix Computation with Discrete Fourier Transform

Nitish Mital, Cong Ling, *Member, IEEE*, and Deniz Gündüz, *Fellow, IEEE*

Abstract—**We consider the problem of secure distributed matrix computation (SDMC), where a *user* queries a function of data matrices generated at distributed *source* nodes. We assume the availability of N honest but curious computation servers, which are connected to the sources, the user, and each other through orthogonal and reliable communication links. Our goal is to minimize the amount of data that must be transmitted from the sources to the servers, called the *upload cost*, while guaranteeing that no T colluding servers can learn any information about the source matrices, and the user cannot learn any information beyond the computation result. We first focus on secure distributed matrix multiplication (SDMM), considering two matrices, and propose a novel polynomial coding scheme using the properties of finite field discrete Fourier transform, which achieves an upload cost significantly lower than the existing results in the literature. We then generalize the proposed scheme to include straggler mitigation, and to the multiplication of multiple matrices while keeping the input matrices, the intermediate computation results, as well as the final result secure against any T colluding servers. We also consider a special case, called computation with own data, where the data matrices used for computation belong to the user. In this case, we drop the security requirement against the user, and show that the proposed scheme achieves the minimal upload cost. We then propose methods for performing other common matrix computations securely on distributed servers, including changing the parameters of secret sharing, matrix transpose, matrix exponentiation, solving a linear system, and matrix inversion, which are then used to show how arbitrary matrix polynomials can be computed securely on distributed servers using the proposed procedure.**

I. INTRODUCTION

In the era of big data, performing computationally intensive operations locally on a single machine is infeasible, and clients often rely on powerful cloud servers to carry out demanding computation tasks. In the so-called *serverless computing paradigm*, clients can request computationally expensive tasks to be performed on massive datasets, potentially generated at multiple geographically distributed locations, using special purpose computing servers (eg., Amazon Web Services (AWS), Microsoft Azure, Google Cloud). While serverless computing provides significant flexibility and speed up, it also leads to growing data privacy concerns, as the corporations

that provide computation services also provide many other digital services, and have access to unprecedented amounts of private user data. Therefore, algorithms that would allow users to benefit from powerful untrustworthy servers while keeping their data private are of significant interest. Our goal in this paper is to design efficient secure distributed matrix computation (SDMC) algorithms, which keep data private from the potentially colluding computing servers as well as the entities requesting the computations.

We consider $\Gamma \geq 1$ data *sources*, represented as matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(\Gamma)}$ on an appropriate finite field. A *user* desires to compute a function of these matrices, $G(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(\Gamma)})$, with the help of N computing servers. The servers are connected to the sources and to each other with orthogonal and reliable links. Similarly, computations carried out by the servers are conveyed to the user over orthogonal and reliable communication links. For a given number of N servers, our goal will be to minimize the amount of data that must be uploaded from the sources to the servers, which we refer to as the *upload cost*. The upload cost often determines the financial cost of serverless computing, but minimizing it would also reduce the overall computing time as it limits the amount of computations that must be carried out by the servers, as well as the communication latency from the sources to the servers, which may be prohibitive especially when the data sources are geographically distant from the servers. For example, the source nodes may be distant hospitals sharing medical data of patients, and the user may be a research institute or a pharmaceutical company making certain queries on the data. In addition to correct computation of the request, we also want to guarantee the privacy of the input data against the servers as well as the requesting user. We impose information theoretic perfect privacy guarantees such that any T colluding servers must not learn anything about the data sources, or the user must not learn anything about the data sources apart from the computation result. We assume that all the servers are honest and responsive, but curious, which means that they follow the prescribed protocol honestly, but any T of them may collude to try to deduce information about the input matrices. We will also consider the special setting of *computation with own data*, in which case the user wants to compute a function on its own data matrices using the available computing servers. In this case we drop the privacy requirement against the user, and the problem lends itself to further optimization.

We will first focus on the secure distributed matrix multiplication (SDMM) problem, which has received significant recent interest. Large scale matrix multiplication is a fundamental building block of matrix computations in many machine

The authors are with the Department of Electrical and Electronics Engineering, Imperial College London.

learning, optimization, and signal processing algorithms. It is also one of the most computationally intensive operations. Moreover, it can be easily distributed across multiple servers thanks to its inherently parallel structure. We will first consider the multiplication of $\Gamma = 2$ matrices, which will allow us to introduce the main ideas behind our design. We then extend our analysis to the multiplication of multiple matrices, as well as to other fundamental matrix operations, which, when combined with matrix multiplication, allow computation of arbitrary polynomials of matrices.

A. Related Work

The cryptography community has extensively studied the problem of secure multi-party computation (MPC), also known as secure function evaluation, in which Alice and Bob, having inputs x and y, respectively, want to compute a function $f(x, y)$ jointly, without any of them learning anything about the other's input either from the communication, or from the result of the computation [1]. The SDMM problem is related to MPC yet different; the design has to ensure that no computing server learns anything about the original data, but we can decide which part of the data is revealed to each server and in what form. Fully homomorphic encryption (FHE) is a class of cryptographic schemes that allow computations on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on plaintext. These techniques rely on working over polynomial rings, and their security is based on the assumed (or proven) hardness of problems in ideal lattices [2], [3]. However, existing FHE schemes are slow and impractical. "Somewhat homomorphic encryption" (SHE) has been proposed as an alternative, which allows a limited number of homomorphic operations on ciphertexts. SHE is relatively faster, and ciphertext packing methods have been proposed for operations like secure inner products [2], [4], and secure matrix multiplications [3], which generalizes Yasuda et al.'s packing method for inner products in [4]. Some works also propose methods for performing other matrix computations in an information theoretically secure manner, like Gaussian elimination, matrix inversion, comparison, equality test, or exponentiation [5], [6].

There is also a growing literature on distributed matrix multiplication, where a lot of effort has been put into speeding up computations, increasing reliability, and/or reducing communication overhead using coding and communication theoretic ideas [7]–[13]. The initial papers considered a slightly different context of speeding up parallel computations by introducing "computation redundancy" to mitigate the problem of straggling servers [7], [8]. Straggling servers refer to slow/unresponsive servers due to which completion of the computation is delayed. A standard way of dealing with stragglers is to introduce "computation redundancy", that is, assigning extra computations to each server. In coded computation against stragglers, the performance metric is the *recovery threshold*; that is, the minimum integer r such that the computation result is recoverable from any r successful (non-delayed, non-faulty) servers. The common theme in the 'coded

computation' literature is to treat stragglers as 'erasures' in communications, and exploit ideas for coding against erasures, which allows reliable reconstruction of the desired result from an arbitrary set of successfully received symbols. Reference [9] uses polynomial codes to construct a scheme in which the computation is completed as long as any K out of N evaluations of a polynomial are received from the servers. To multiply two matrices $\mathbf{A}$ and $\mathbf{B}$ with the help of N servers, the polynomial code in [9] partitions $\mathbf{A}$ row-wise and $\mathbf{B}$ column-wise (row-by-column partitioning), and generates encoded matrices as evaluations of a polynomial with the blocks as the coefficients, similarly to Reed Solomon codes [14]. A follow-up work [10] proposes a new polynomial coded computation scheme called MatDot, which achieves the optimal recovery threshold for column-wise partitioning of matrix $\mathbf{A}$ and row-wise partitioning of $\mathbf{B}$ (sum-of-outer-products method), which we shall refer to as the column-by-row partitioning henceforth in the paper, at the expense of an increase in the communication cost. In [10], the authors also propose PolyDot codes that interpolate between the polynomial codes of [9] and MatDot codes. PolyDot codes are later improved as generalized PolyDot (GPD) codes in [11], which achieve the optimal recovery threshold for any arbitrary partitioning of the input matrices. Entangled polynomial codes, proposed in parallel in [12], also achieve the same performance as GPD codes. Bivariate polynomial codes are introduced in [15] for straggler mitigation when servers can compute and transmit multiple partial computations.

Subsequent papers, inspired by the works on straggler mitigation in distributed matrix multiplication, consider the SDMM problem from an information theoretic perspective. These papers aim for information theoretic security, independent of the computational capacities of the attackers, as opposed to cryptographic techniques. These works typically assume that the data to be used for computations belongs to the user; hence, they focus on privacy against the servers. The earlier papers on SDMM consider *download rate* as the performance metric, which is defined as the ratio of the number of bits required to represent the computation result to the total number of bits that the servers must transmit to the user. Reference [16] uses the idea of polynomial codes from the literature on straggler mitigation to propose an SDMM scheme based on Shamir's secret sharing scheme [17], which is shown to achieve the optimal download rate for *one-sided* SDMM (where only one of the matrices is kept secure). An achievable scheme is also proposed for *two-sided* SDMM (both matrices are kept secure). Reference [18] introduces GASP codes, which improve the download rate for two-sided SDMM by aligning the degrees of the terms in the polynomial code so that the desired products appear as distinct terms.

In [19], new converse bounds on the optimal download rate for SDMM are obtained by showing that the capacity of a multi-message X-secure T-private information retrieval (MM-XSTPIR) problem ([20], [21]) provides an upper bound on the download rate of the SDMM problem. The optimal download rate of the MM-XSTPIR problem is shown to depend on the dimensions of the matrices $\mathbf{A}$ and $\mathbf{B}$. The scheme in [19] allows the joint retrieval of a batch of matrix products (*batch

multiplication), instead of multiplying two matrices using the *matrix partitioning* approach, resulting in a coding gain. Other recent works on secure distributed batch matrix multiplication include [22] and [23].

Reference [13] combines straggler mitigation and secure computation using the batch multiplication approach. Byzantine security is also considered, which refers to security against adversarial servers that may actively corrupt the results they send back to the user. Their scheme is based on Lagrange polynomials, and it achieves the optimal recovery threshold for any multi-linear function computation. Since the scheme in [13] is designed for batch computation of any function, we can adapt it for matrix multiplication using a matrix-partitioning based approach, and compare its performance with that of other matrix partitioning based schemes. This can be done by treating the partitions of the matrices to be multiplied as batches of data. Therefore, if $\mathbf{A}$ is partitioned column-wise and $\mathbf{B}$ is partitioned row-wise into K partitions each, the scheme in [13] has the recovery threshold of $2(K+T-1)+S+2A+1$, where S and A are the numbers of stragglers and Byzantine adversaries, respectively. If $\mathbf{A}$ is partitioned row-wise and $\mathbf{B}$ is partitioned column-wise, into K and L partitions respectively, the recovery threshold is $2(KL+T-1)+S+2A+1$, which is the same performance as that of the GASP codes in [18] for big T, which can be seen by setting $S = A = 0$. For arbitrary matrix partitions, the Lagrange coded scheme does not perform as well as the secure generalized PolyDot (SGPD) codes introduced later in [24]. In [24], the trade-off between the download rate and the recovery threshold, first studied in [10] using the MatDot and PolyDot schemes for straggler mitigation, is extended to SDMM. The SGPD codes achieve the same recovery threshold as Lagrange codes for the sum-of-outer-products method.

In [25], the trade-off between the upload and download costs for SDMM is studied. While the *download cost* is simply the reciprocal of the download rate, the *upload cost* is defined as the ratio of the total number of bits that the user must send to the servers to the total size in bits of the data matrices. In [25], a secure cross subspace alignment (SCSA) scheme with adjustable upload cost (USCSA) is presented for SDMM. The tradeoff between the upload and download costs for SDMM schemes using only the row-by-column partitioning is studied in [25], but not the tradeoff for SDMM schemes using the column-by-row partitioning.

The work that is most related to ours is [26], which considers the setting in which the data is generated at distributed source nodes, and does not belong to the user requesting the computation. It extends the BGW (Ben-Or, Goldwasser and Widgerson) scheme from [27], which was first proposed in the context of secure MPC, for multiplication of matrices using a connected network of computing servers. The sources are assumed not to be connected with each other, while the servers are. The latter assumption is exploited to reduce the communication to the user by allowing the servers to cooperate securely. Thus, the servers share their results from the first round of computation among each other using Shamir's secret sharing scheme, and compute a linear combination of the received shares of the results. This inter-server cooperation

allows a smaller number of servers to send these linear combinations to the user. This particular model also imposes privacy against the user, that is, the user cannot learn anything about the data beyond what it learns from the computation result. This constraint is not imposed in other papers, where the user is the source of the input data; and neither in [19], where distributed source nodes generate the data.

B. Main contributions:

With respect to the rich literature on the topic that we have summarized above, the main novel contributions of our work can be summarized as follows:

- We first introduce a novel polynomial coding scheme exploiting the properties of discrete Fourier transform, and show that it achieves a near optimal upload cost for SDMM of two matrices, while achieving the optimal upload cost for the special case of computation with own data, in which the user has access to the matrices used in the computations. The key difference between the existing polynomial coded SDMM schemes and the one proposed in this paper is that instead of evaluating the polynomial over some arbitrary distinct points, here we evaluate the polynomial at the N-th roots of unity, which yields a discrete Fourier transform over the finite field, also known as the number theoretic transform (NTT).
- The proposed scheme can be implemented in an efficient manner using the recently developed fast Fourier transform (FFT) algorithm on finite fields [28], and has negligible decoding complexity.
- We generalize the proposed scheme for SDMM of two matrices to introduce straggler mitigation.
- We extend the proposed scheme to securely multiply multiple matrices on distributed servers securely. Our scheme has a significantly lower upload cost than the existing alternatives, and is naturally scalable to the multiplication of an arbitrary number of matrices.
- We also present some schemes for other matrix operations, such as addition, transpose, exponentiation, changing the parameters of the secret shares, and solving linear systems, which includes computing the matrix inverse. As a result, it is shown that arbitrary matrix polynomials can be computed securely on distributed servers.

Notations: The notation $[b]$ denotes the set of consecutive integers $\{1, \ldots, b\}$. The set of natural numbers is denoted by $\mathbb{N}$. Sets are denoted by calligraphic letters $\mathcal{L}$. Matrices and vectors are denoted by bold upper-case letters $\mathbf{A}$ and bold lower-case letters $\mathbf{a}$, respectively. The notation $[[\mathbf{A}]]_i$ denotes the secret share of matrix $\mathbf{A}$ delivered to server i, while the notation $[[\mathbf{A}]]_{\mathcal{L}}$ denotes the set of secret shares of matrix $\mathbf{A}$ delivered to servers belonging to the set $\mathcal{L}$.

II. SYSTEM MODEL

We consider Γ source nodes, $N \geq 2$ servers, and one user, for some $\Gamma, N \in \mathbb{N}$ (see Fig. 1). Each source node is connected to each server through an orthogonal link. Each pair of servers is connected to each other, and each server is connected to the user through a private link. Each source

node $\gamma \in [\Gamma] \triangleq \{1, \ldots, \Gamma\}$ has access to an input matrix $\mathbf{A}^{(\gamma)} \in \mathbb{F}_q^{m_\gamma \times m'_\gamma}$, for $m_\gamma, m'_\gamma \in \mathbb{N}, \gamma \in [\Gamma]$, and a finite field $\mathbb{F}_q$ with q elements. Using N computing servers, the user wants to securely compute the result of a function $\mathbf{C} = G(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(\Gamma)})$, where G is an arbitrary polynomial function, assuming appropriate matrix dimensions. We assume that the entries of $\mathbf{A}^{(\gamma)}, \gamma \in [\Gamma]$, are independent of each other and uniformly distributed over $\mathbb{F}_q$. We also assume that the servers are honest, but curious, which means that each server honestly follows the protocol without spurious insertions, yet may infer information about the inputs passively. Similarly to [26], the system operates in three phases: (1) Sharing, (2) Computation and communication, and (3) Reconstruction. A detailed description of these phases is as follows.

1) **Sharing phase:** In this phase, the source γ sends secret shares of matrix $\mathbf{A}^{(\gamma)}$, denoted by $[[\mathbf{A}^{(\gamma)}]]_i$, to server i. $[[\mathbf{A}^{(\gamma)}]]_i$ is a function of input matrix $\mathbf{A}^{(\gamma)}$ and a secret key $\mathbf{Sk}^{(\gamma)}$, and its dimensions will depend on the computing scheme.

2) **Computation and communication:** In this phase, the servers process the data they have received from the sources, and may also exchange messages with each other. We denote the set of all messages that server i sends to server i' in this phase by $\mathcal{M}_{i,i'}$.

3) **Reconstruction:** In this phase, every server $i \in [N]$ sends a message $[[\mathbf{C}]]_i$ to the user, who decodes the received messages to recover the desired computation.

The scheme must satisfy the following four constraints.

a) Correctness: The user must be able to decode the final function $\mathbf{C} = G(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(\Gamma)})$ from the responses it receives from the servers, $[[\mathbf{C}]]_1, \ldots, [[\mathbf{C}]]_N$. The correctness constraint is imposed by:

$$H\left(\mathbf{C}\big|[[\mathbf{C}]]_1, \ldots, [[\mathbf{C}]]_N\right) = 0. \tag{1}$$

Remark 1. *We will also consider the special case of computation with own data, where the data matrices used for computation belong to the user, as in the distributed computation scenario studied in [13], [16], [18], [24]. In this scenario, the correctness constraint becomes:*

$$H\left(\mathbf{C}\big|[[\mathbf{C}]]_1, \ldots, [[\mathbf{C}]]_N, \mathbf{Sk}^{(1)}, \ldots, \mathbf{Sk}^{(\Gamma)}\right) = 0. \tag{2}$$

b) Security against colluding servers: The goal is to recover $\mathbf{C}$ reliably and securely even if any $T < N$ servers collude to extract some information about the input matrices. Hence, for any $\mathcal{L} \subset [N]$ with $|\mathcal{L}| \leq T$, the encoded matrices $[[\mathbf{A}^{(\gamma)}]]_\mathcal{L}$, and the set of messages communicated by the servers in $\mathcal{L}^c$ to the servers in $\mathcal{L}$, denoted by $\mathcal{M}_{\mathcal{L}^c, \mathcal{L}}$, must not reveal any information about the source matrices, $\{\mathbf{A}^{(\gamma)}\}_{\gamma=1}^\Gamma$. Accordingly, the security constraint is specified as,

$$I\left(\left\{\mathbf{A}^{(\gamma)}\right\}_{\gamma=1}^\Gamma ; \left\{[[\mathbf{A}^{(\gamma)}]]_\mathcal{L}\right\}_{\gamma=1}^\Gamma, \mathcal{M}_{\mathcal{L}^c, \mathcal{L}}\right) = 0,$$
$$\forall \mathcal{L} \subseteq [N], |\mathcal{L}| \leq T. \tag{3}$$

c) Security against the user: The user must not gain additional information about the input matrices beyond the result of the function G. This is defined as:

$$I\left(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(\Gamma)}; [[\mathbf{C}]]_1, \ldots, [[\mathbf{C}]]_N \big| \mathbf{C}\right) = 0.$$

For a given number of servers N and security requirement T, the performance will be measured in terms of the *upload cost* from the source nodes to the servers. The upload cost is defined as follows:

$$\chi_{UL} \triangleq \frac{\sum_{\gamma=1}^\Gamma \sum_{i=1}^N H([[\mathbf{A}^{(\gamma)}]]_i)}{\sum_{\gamma=1}^\Gamma H(\mathbf{A}^{(\gamma)})}, \tag{4}$$

and it quantifies the normalized amount of information that must be delivered to the servers. In most cases the amount of computation that must be carried out by each server depends on the amount of information delivered to it, and many cloud computing services charge users based on the amount of information delivered and stored at each server. Hence, minimizing the upload cost will reduce both the latency and the cost of computations. Accordingly, our objective is to securely compute $\mathbf{C}$ by incurring the minimum upload cost. Due to the symmetry across the servers, we will assume (unless stated otherwise) that the secret shares sent from each source to each server is of the same size, and is a $(1/K)^{th}, K \in \mathbb{N}$, fraction of the size of the input matrices.

Definition 1. *An (N, K, T) SDMC scheme uses N servers, sends $(1/K)^{th}$ fraction of the input data matrices' size to each server, and is secure against any T colluding servers. Hence, the upload cost of an (N, K, T) SDMC scheme is given by $\chi_{UL} = \frac{N}{K}$.*

III. SDMM FOR $\Gamma = 2$

In this section we focus exclusively on the secure distributed multiplication of two matrices, i.e., $\Gamma = 2$. This problem will allow us to present the main ideas behind our coded computation scheme. We first present our main result in the following theorem.

Theorem 2. *We can securely multiply two matrices using N servers, T of which may collude, with $N > 2T$, with an upload cost of $\frac{N}{N-2T}$. In other words, an $(N, N-2T, T)$ SDMM scheme for two input matrices is achievable.*

We present the following example to illustrate the essential ingredients of the scheme. For ease of notation, we use the notation $\mathbf{A}^{(1)} = \mathbf{A}$ and $\mathbf{A}^{(2)} = \mathbf{B}$ in this section.

Example: We consider distributed multiplication of matrices $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ and $\mathbf{B} \in \mathbb{F}_q^{n \times p}$ over $N = 7$ servers, any $T = 2$ of which may collude (see Fig. 1).

1) Sharing phase: The matrices $\mathbf{A}$ and $\mathbf{B}$ are partitioned into $K = N - 2T = 3$ blocks of dimensions $m \times \frac{n}{3}$ and $\frac{n}{3} \times p$, respectively, as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \mathbf{B}_3 \end{bmatrix}, \tag{5}$$

and the product $\mathbf{C} = \mathbf{A}\mathbf{B}$ is given by

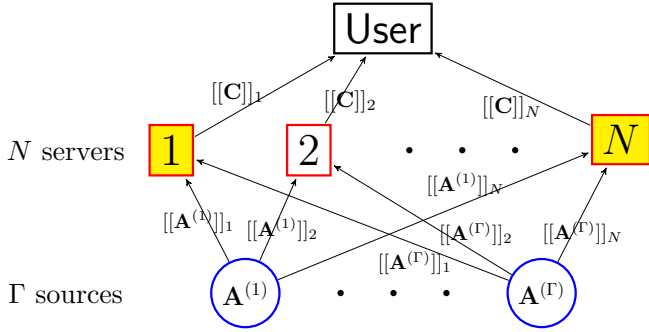$$\mathbf{C} = \mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_2 + \mathbf{A}_3\mathbf{B}_3. \tag{6}$$

Fig. 1: System model for Γ matrices with N servers, any T of which may collude (shown as shaded servers).

The matrices $\mathbf{R}_i \in \mathbb{F}_q^{m \times \frac{n}{K}}$ and $\mathbf{S}_i \in \mathbb{F}_q^{\frac{n}{K} \times p}, i \in [T]$, are generated, whose entries are independent and identically distributed (i.i.d.) uniform random variables from the finite field $\mathbb{F}_q$. The matrices $\mathbf{R} = [\mathbf{R}_1, \ldots, \mathbf{R}_T]$ and $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_T]$ are used as the secret keys to encode the input matrices $\mathbf{A}$ and $\mathbf{B}$, respectively. The following two polynomials are constructed to encode the matrices:

$$\mathbf{A}(x) = \mathbf{A}_1 + \mathbf{A}_2 x + \mathbf{A}_3 x^2 + \mathbf{R}_1 x^3 + \mathbf{R}_2 x^4 \qquad (7)$$

$$\mathbf{B}(x) = \mathbf{B}_1 + \mathbf{B}_2 x^{-1} + \mathbf{B}_3 x^{-2} + \mathbf{S}_1 x^{-5} + \mathbf{S}_2 x^{-6}. \qquad (8)$$

Note that the encoding polynomials of $\mathbf{A}$ and $\mathbf{B}$ are different from each other, depending on the order of multiplication. We refer to the encoding of matrix $\mathbf{A}$ as "left-encoding", and that of matrix $\mathbf{B}$ as "right-encoding". We refer to each of the N secret shares obtained from left-encoding of matrix $\mathbf{A}$ as an "(N, K, T) left-share", where the $i^{th}, \forall i \in [N]$ secret share is denoted by $[[\mathbf{A}]]_i^L$, and each of those from right-encoding of matrix $\mathbf{B}$ as an "(N, K, T) right-share", where the $i^{th}, \forall i \in [N]$ secret share is denoted by $[[\mathbf{B}]]_i^R$.

Let α_7 be a primitive 7th root of unity in $\mathbb{F}_q$, and $1, \alpha_7, \alpha_7^2, \ldots, \alpha_7^6$ be the 7th roots of unity in $\mathbb{F}_q$. Then the polynomials $\mathbf{A}(x)$ and $\mathbf{B}(x)$ are evaluated at α_7^{i-1} to obtain $[[\mathbf{A}]]_i^L, [[\mathbf{B}]]_i^R$; that is, $[[\mathbf{A}]]_i^L = \mathbf{A}(\alpha_7^{i-1})$ and $[[\mathbf{B}]]_i^R = \mathbf{B}(\alpha_7^{i-1}), i = 1, \ldots, 7$, and the secret shares $[[\mathbf{A}]]_i^L, [[\mathbf{B}]]_i^R$ are sent to server $i, i = 1, \ldots, 7$. The number of symbols sent by the source nodes to the servers is given by $H([[\mathbf{A}]]_i) + H([[\mathbf{B}]]_i) = \frac{mn}{K} + \frac{np}{K}$.

Remark 2. *We note here that evaluating the polynomials $\mathbf{A}(x)$ and $\mathbf{B}(x)$ at the roots of unity is equivalent to computing the discrete Fourier transform of the sequences $\{\mathbf{A}_1, \ldots, \mathbf{A}_K, \mathbf{R}_1, \ldots, \mathbf{R}_T\}$ and $\{\mathbf{B}_1, \ldots, \mathbf{B}_K, \mathbf{0}_1, \ldots, \mathbf{0}_T, \mathbf{S}_1, \ldots, \mathbf{S}_T\}$ in a finite field, where $\mathbf{0}_k, k = 1, \ldots, T$, are zero matrices. This can be carried out efficiently using the FFT algorithm for finite fields introduced in [28].*

2) Computation phase: Server i computes the product

$$[[\mathbf{C}]]_i = [[\mathbf{AB}]]_i = [[\mathbf{A}]]_i^L [[\mathbf{B}]]_i^R, \qquad (9)$$

which is equivalent to the evaluation of the polynomial

$$\mathbf{C}(x) = \mathbf{A}(x)\mathbf{B}(x) \qquad (10)$$

$$= \mathbf{A}_1 \mathbf{S}_2 x^{-6} + (\mathbf{A}_1 \mathbf{S}_1 + \mathbf{A}_2 \mathbf{S}_2) x^{-5} +$$

$$\cdots + (\mathbf{A}_1 \mathbf{B}_2 + \mathbf{A}_2 \mathbf{B}_3 + \mathbf{R}_2 \mathbf{S}_1) x^{-1}$$

$$+ \left(\sum_{l=1}^{3} \mathbf{A}_l \mathbf{B}_l \right) + \cdots + \mathbf{R}_2 \mathbf{B}_1 x^4 \qquad (11)$$

$$= \sum_{l=1}^{3} \mathbf{A}_l \mathbf{B}_l + (\mathbf{A}_2 \mathbf{B}_1 + \mathbf{A}_1 \mathbf{S}_2 + \mathbf{A}_3 \mathbf{B}_2 + \mathbf{R}_1 \mathbf{B}_3) x$$

$$+ \cdots + (\mathbf{A}_3 \mathbf{S}_1 + \mathbf{R}_1 \mathbf{S}_2 + \mathbf{R}_2 \mathbf{B}_1) x^4 +$$

$$\cdots + (\mathbf{A}_1 \mathbf{B}_2 + \mathbf{A}_2 \mathbf{B}_3 + \mathbf{R}_2 \mathbf{S}_1) x^6 \quad (12)$$

at $x = \alpha_7^{i-1}$.

Remark 3. *If we evaluate the polynomial over some arbitrary distinct points, then the product of two secret shares corresponds to the product of two polynomials, which is equivalent to the so-called linear convolution. On the other hand, if we evaluate the polynomial over the N-th roots of unity, as is done in our paper, then the multiplication of two secret shares corresponds to the product of two polynomials modulo $x^N - 1$, which is equivalent to the so-called circular convolution. While the circular convolution introduces aliasing, since the only desired term for inner-product type of matrix multiplication is the constant term (i.e., the DC term), the correctness of the result is guaranteed despite aliasing. Such aliasing allows further interference alignment, which improves the efficiency compared to linear convolution.*

3) Reconstruction phase: The servers send $[[\mathbf{C}]]_i$'s to the user. We know that

$$\sum_{i=1}^{N} (\alpha_N^{i-1})^s = 0, \qquad \forall s : N \nmid s. \qquad (13)$$

Therefore, the user computes the average of the received responses to obtain the final result:

$$\frac{1}{7} \sum_{i=1}^{7} [[\mathbf{C}]]_i = \sum_{l=1}^{3} \mathbf{A}_l \mathbf{B}_l, \qquad (14)$$

because the non-constant terms from Eq. (11) sum to 0 thanks to Eq. (13).

For the general case with N servers, T of which can collude, the input matrices are partitioned into $K = N - 2T$ submatrices similarly to Eq. (5). Then the product can be written as $\mathbf{C} = \mathbf{AB} = \sum_{l=1}^{K} \mathbf{A}_l \mathbf{B}_l$.

The matrices $\mathbf{A}$ and $\mathbf{B}$ are encoded with the following two polynomials:

$$\mathbf{A}(x) = \sum_{l=1}^{K} \mathbf{A}_l x^{l-1} + \sum_{l=1}^{T} \mathbf{R}_l x^{K+l-1}, \qquad (15)$$

and

$$\mathbf{B}(x) = \sum_{l=1}^{K} \mathbf{B}_l x^{-l+1} + \sum_{l=1}^{T} \mathbf{S}_l x^{-K-T-l+1}. \qquad (16)$$

We define the product polynomial as follows.

$$\mathbf{C}(x) = \sum_{l=1}^{K} \mathbf{A}_l \mathbf{B}_l + \text{(non-constant terms)}. \qquad (17)$$

The goal is to recover the constant term in $\mathbf{C}(x)$ from the computations of N servers. The polynomials $\mathbf{A}(x)$ and $\mathbf{B}(x)$ are evaluated at the Nth roots of unity, denoted by $1, \alpha_N, \alpha_N^2, \ldots, \alpha_N^{N-1} \in \mathbb{F}_q$, where α_N is a primitive N^{th} root of unity in $\mathbb{F}_q$. Thus, the values sent to server i are $[[\mathbf{A}]]_i^L = \mathbf{A}(\alpha_N^{i-1})$ and $[[\mathbf{B}]]_i^R = \mathbf{B}(\alpha_N^{i-1})$. Server $i \in [N]$ computes the share $[[\mathbf{C}]]_i = [[\mathbf{A}]]_i^L [[\mathbf{B}]]_i^R = \mathbf{C}(\alpha_N^{i-1})$, and sends it to the user.

Thanks to Eq. (13), the user obtains the desired result by averaging the received $[[\mathbf{C}]]_i$'s.

$$\frac{1}{N} \sum_{i=1}^{N} [[\mathbf{C}]]_i = \sum_{l=1}^{K} \mathbf{A}_l \mathbf{B}_l. \qquad (18)$$

Remark 4. *Since the polynomials must be evaluated at the Nth roots of unity in the proposed scheme, the finite field must be chosen to guarantee the presence of all the Nth order roots of unity. Therefore, we must have $N|(q-1)$. This can be satisfied by appropriately choosing the field size q. This also guarantees that the multiplicative inverse of N exists in $\mathbb{F}_q$, so that the $\frac{1}{N}$ in Eq. (18) exists.*

A. Special case: Computation with own data

For the special case in which the data matrices belong to the user, the upload cost can be further reduced since the user also has access to the random secret keys used for generating the secret shares. Most previous literature on SDMM considers this special case. The user partitions the input matrices into K blocks as in Eq. (5), where $K = N - T$. The user then encodes the matrices using the following polynomials.

$$\mathbf{A}(x) = \sum_{l=1}^{K} \mathbf{A}_l x^{l-1} + \sum_{l=1}^{T} \mathbf{R}_l x^{K+l-1}, \qquad (19)$$

and

$$\mathbf{B}(x) = \sum_{l=1}^{K} \mathbf{B}_l x^{-l+1} + \sum_{l=1}^{T} \mathbf{S}_l x^{-K-l+1}, \qquad (20)$$

where $\mathbf{A}(x)$ is the same as in Eq. (15) while we have a slight change from Eq. (16) in the way the secret key is embedded into the $\mathbf{B}(x)$ polynomial. The user evaluates the polynomials on the Nth roots of unity to generate the secret shares, and sends these shares to the servers. The servers return their computed results back to the user, where the constant term in the product polynomial $\mathbf{C}(x)$ is now given by $\sum_{i=1}^{K} \mathbf{A}_i \mathbf{B}_i + \sum_{i=1}^{T} \mathbf{R}_i \mathbf{S}_i$. The user then averages the received results to obtain

$$\frac{1}{N} \sum_{i=1}^{N} [[\mathbf{C}]]_i = \sum_{l=1}^{K} \mathbf{A}_l \mathbf{B}_l + \sum_{l=1}^{T} \mathbf{R}_l \mathbf{S}_l. \qquad (21)$$

Since the user has access to the secret keys it has used to encrypt the partitions, it can subtract their product, $\sum_{i=1}^{T} \mathbf{R}_i \mathbf{S}_i$, to obtain the desired result. We note here that the product of the random matrices $\mathbf{R}$ and $\mathbf{S}$ needs to be pre-computed by the user in order to obtain the desired matrix product from Eq. (21). If $T << N - T$, this would require much less computational resources compared to multiplying the matrices $\mathbf{A}$ and $\mathbf{B}$, or alternatively these multiplications can be done in advance in an offline manner, and stored at the user, and hence, this computation does not affect the computation latency.

Theorem 3. *A $(N, N - T, T)$ SDMM scheme for two input matrices is achievable for the special case, or in other words, we can securely multiply two matrices using N servers, T of which may collude, with $N > T$, with an optimal upload cost of $\frac{N}{N-T}$.*

Proof. It follows from the definition of an SDMM scheme that the upload cost of the $(N, N-T, T)$ SDMM scheme proposed above is $\chi_{UL} = \frac{N}{N-T}$. Moreover, it is proved in [25] that the optimal upload cost of an SDMM scheme is lower bounded by $\frac{N}{N-T}$. This proves the optimality of the proposed scheme in terms of the upload cost. $\qquad \square$

B. Proof of security against colluding servers

We next prove that the proposed scheme is secure, i.e., the security constraint (3) is satisfied. We point out that there is no exchange of messages between the servers, that is, $\mathcal{M}_{i,i'} = \phi$ for all $i, i' \in [N], i \neq i'$. For any $\mathcal{L}$ with $|\mathcal{L}| = T$, we have

$$I(\mathbf{A}, \mathbf{B}; [[\mathbf{A}]]_\mathcal{L}, [[\mathbf{B}]]_\mathcal{L}) \qquad (22)$$
$$= H([[\mathbf{A}]]_\mathcal{L}, [[\mathbf{B}]]_\mathcal{L}) - H([[\mathbf{A}]]_\mathcal{L}, [[\mathbf{B}]]_\mathcal{L} | \mathbf{A}, \mathbf{B}) \qquad (23)$$
$$\overset{a}{=} H([[\mathbf{A}]]_\mathcal{L}, [[\mathbf{B}]]_\mathcal{L}) - H(\mathbf{R}, \mathbf{S}) \qquad (24)$$
$$\overset{b}{=} H([[\mathbf{A}]]_\mathcal{L}) + H([[\mathbf{B}]]_\mathcal{L}) - H(\mathbf{R}) - H(\mathbf{S}) \qquad (25)$$
$$\leq \sum_{i \in \mathcal{L}} H([[\mathbf{A}]]_i) + \sum_{i \in \mathcal{L}} H([[\mathbf{B}]]_i) - \frac{mnT}{K} \log |\mathbb{F}_q|$$
$$\qquad\qquad\qquad - \frac{pnT}{K} \log |\mathbb{F}_q| \qquad (26)$$
$$= 0, \qquad (27)$$

where (a) follows from (15) and (16); (b) follows from the fact that $[[\mathbf{A}]]_\mathcal{L}$, $[[\mathbf{B}]]_\mathcal{L}$, $\mathbf{R}$ and $\mathbf{S}$ are independent of each other; (27) follows because the elements of the secret shares $[[\mathbf{A}]]_i \in \mathbb{F}_q^{m \times \frac{n}{K}}$ and $[[\mathbf{B}]]_i \in \mathbb{F}_q^{\frac{n}{K} \times p}, \forall i \in \mathcal{L}$ are independent and uniformly distributed in $\mathbb{F}_q$. Hence, $\sum_{i \in \mathcal{L}} H([[\mathbf{A}]]_i) = \frac{mnT}{K} \log |\mathbb{F}_q|$ and $\sum_{i \in \mathcal{L}} H([[\mathbf{B}]]_i) = \frac{pnT}{K} \log |\mathbb{F}_q|$, where $|\mathbb{F}_q|$ denotes the cardinality of the field $\mathbb{F}_q$.

C. Security against the user

When the input matrices are generated by distributed source nodes, the user must not gain additional information beyond the result of the computation. Most existing schemes [13], [16], [18], [24] which rely on polynomial interpolation, do not satisfy this condition, since they are designed for the case when the data is generated by the user. If those schemes are employed, once $\mathbf{C}(x) = \mathbf{A}(x)\mathbf{B}(x)$ is interpolated by the user after receiving the evaluations of $\mathbf{C}(x)$ at a number of points equal to the number of terms in $\mathbf{C}(x)$, it can be factorized to obtain information about $\mathbf{A}(x)$ and $\mathbf{B}(x)$, thus leaking additional information to the user. Generally, factorizations of

matrices are not unique in any field, but even then they can narrow the search space significantly in certain cases, thus leaking partial information, and in some cases, factorization leaks complete information of $\mathbf{A}$ and $\mathbf{B}$ by providing unique factors. For example, consider that $m = p = 1$ and $K = n$, and the polynomials $\mathbf{A}(x)$ and $\mathbf{B}(x)$ are irreducible polynomials over $\mathbb{F}_q$. Then, on recovering the polynomial $\mathbf{C}(x) = \mathbf{A}(x)\mathbf{B}(x)$, there is a unique factorization of $\mathbf{C}(x)$ in $\mathbb{F}_q$ which provides the factors as the two constituent irreducible polynomials. Our scheme is robust to such information leakages.

We point out that besides the constant term, the user can recover the sum of the coefficients of x^{N-i} and $x^{-i}, i \in [N-1]$, due to aliasing in Eq. (11), because $\alpha_N^{-i} = \alpha_N^{N-i}$. The polynomial after considering the aliasing effect is shown in Eq. (12). These residual terms may or may not leak partial information about the input matrices to the user. To show this, first we mention a result from [19] that determines the entropy of the product of two matrices depending on their dimensions:

Lemma 4. *Let* $\mathbf{P}, \mathbf{Q}$ *be random matrices independently and uniformly distributed over* $\mathbb{F}_q^{m \times n}$ *and* $\mathbb{F}_q^{n \times p}$, *respectively. As* $q \to \infty$, *we have*

$$H(\mathbf{PQ}) = \begin{cases} mp & n \geq \min(m,p) \\ mn + np - n^2 & n < \min(m,p) \end{cases}, \quad (28)$$

$$H(\mathbf{PQ} \mid \mathbf{P}) = \min\{mp, np\} \quad (29)$$

$$H(\mathbf{PQ} \mid \mathbf{Q}) = \min\{mp, mn\} \quad (30)$$

in $q-ary$ *units.*

A relaxed notion of security against the user can be shown to be satisfied under certain conditions. The coefficient of each power of x, where the coefficient of the jth power of x is denoted by $\mathbf{C}_j$, is the sum of at least K matrix products. Therefore, the coefficient of x^4, for example, can be written as:

$$\mathbf{C}_4 = \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1 = \begin{bmatrix} \mathbf{A}_3 & \mathbf{R}_1 & \mathbf{R}_2 \end{bmatrix} \begin{bmatrix} \mathbf{S}_1 \\ \mathbf{S}_2 \\ \mathbf{B}_1 \end{bmatrix},$$
$$(31)$$

which is a product of two matrices of dimensions $m \times n$ and $n \times p$, respectively. If $n \geq \max(m,p)$, and matrices $\mathbf{A}$ and $\mathbf{B}$ are i.i.d. uniform, then from Lemma 4 we have $H(\mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = H(\mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1 \mid \mathbf{A}, \mathbf{R}) = mp$ in q-ary units. Therefore, we have $I(\mathbf{A}, \mathbf{R}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = 0$. Similarly, we have $I(\mathbf{B}, \mathbf{S}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = 0$. Thus the coefficient of x^4, that is $\mathbf{C}_4$, is i.i.d. uniformly distributed and independent of the input matrices. Similarly, we have $I(\mathbf{A}, \mathbf{R}; \mathbf{C}_j) = 0$ and $I(\mathbf{B}, \mathbf{S}; \mathbf{C}_j) = 0$ for all $j = 1, \ldots, N-1$. On the other hand, if $p \leq n \leq m$, we have $I(\mathbf{A}, \mathbf{R}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = mp - np > 0$, and $I(\mathbf{B}, \mathbf{S}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = 0$, while if $m \leq n \leq p$, we have $I(\mathbf{A}, \mathbf{R}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = 0$, and $I(\mathbf{B}, \mathbf{S}; \mathbf{A}_3\mathbf{S}_1 + \mathbf{R}_1\mathbf{S}_2 + \mathbf{R}_2\mathbf{B}_1) = mp - mn > 0$.

When $n < \min(m,p)$, or in general, when matrices $\mathbf{A}$ and $\mathbf{B}$ are not i.i.d. uniform randomly distributed, the user can infer partial information about both $\mathbf{A}$ and $\mathbf{B}$, and instead the following procedure can be implemented where the servers exchange shares of their computed results with each other in a

secure way to discard the residual terms, thus ensuring security against the user. The servers exchange shares of their results from the computation phase in a secure way, similarly to [26], so that each server ends up with a $(N, N - T, T)$ left-share of matrix $\mathbf{C}$, which can then be delivered to the user. To do this,

- Server i generates $(N, N - T, T)$ left-shares of $[[\mathbf{C}]]_i$, evaluated on the Nth roots of unity $\alpha_N^{j-1}, \forall j \in [N]$, with α_N being a primitive N^{th} root of unity in $\mathbb{F}_q$, and enumerated as $[[\mathbf{C}]]_{i,j}^L = [[[[\mathbf{C}]]_i]]_j^L, \forall j \in [N]$.
- Server i sends the left-share $[[\mathbf{C}]]_{i,j}^L$ to server j. The privacy requirement against the servers is satisfied, since any T colluding servers cannot gain any information about server i's share $[[\mathbf{C}]]_i$ from the left-shares received in the communication phase.
- Server j averages the received left-shares $[[\mathbf{C}]]_{i,j}^L, \forall i \in [N]$, to obtain the $(N, N - T, T)$ left-share $[[\mathbf{C}]]_j^L$.

To see the correctness of the above procedure, note that, for given N, K and T values, the secret sharing scheme is linear for both left and right shares; that is, $[[\mathbf{A}]] + [[\mathbf{B}]] = [[\mathbf{A} + \mathbf{B}]]$. Therefore, following from Eq. (18), we have

$$\frac{1}{N}\sum_{i=1}^{N}[[\mathbf{C}]]_{i,j}^L = \frac{1}{N}\sum_{i=1}^{N}[[[[\mathbf{C}]]_i]]_j^L \quad (32)$$

$$= [[\mathbf{C}]]_j^L. \quad (33)$$

D. Other performance metrics

1) Encoding complexity: To compute N evaluations of the matrix polynomial $\mathbf{A}(x)$ on the roots of unity, the source nodes perform $\frac{mn}{K}$ N-point FFTs, which involve $O(\frac{mnN \log N}{K})$ finite field operations in $\mathbb{F}_q$. For $T = 0$, the complexity is $O(mn \log N)$, that is, it is a logarithmic rate of growth with respect to N. The analysis is similar for computing the evaluations of matrix polynomial $\mathbf{B}(x)$.

2) Download cost: Download cost is the normalized number of bits that need to be downloaded by the user from the servers to reconstruct the computation result. It is defined as

$$\chi_{DL} = \frac{\sum_{i=1}^{N} H([[\mathbf{C}]]_i)}{H(\mathbf{C})}. \quad (34)$$

As shown in [19], the download cost depends on the dimensions of the data matrices. We can compute the download cost of our scheme for three different cases.

- If $\min(m,p) \leq n \leq K\min(m,p)$, the download cost is

$$\chi_{DL} = \frac{N(\frac{mn}{K} + \frac{np}{K} - \frac{n^2}{K^2})}{mp} \quad (35)$$

$$= \frac{Nn(m + p - \frac{n}{K})}{Kmp}. \quad (36)$$

Assuming $m = p$, we have $\chi_{DL} \approx \frac{2nN}{mK} < \frac{2N}{K}$.
- If $n \geq K\min(m,p)$, the download cost is

$$\chi_{DL} = \frac{Nmp}{mp} \quad (37)$$

$$= N. \quad (38)$$

This article has been accepted for publication in IEEE Transactions on Information Theory. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIT.2022.3158868

8

- If $n < \min(m,p)$, the download cost is

$$\chi_{DL} = \frac{N\left(\frac{mn}{K} + \frac{np}{K} - \frac{n^2}{K^2}\right)}{mn + np - n^2} \qquad (39)$$

$$= \frac{N\left(m + p - \frac{n}{K}\right)}{K(m + p - n)}. \qquad (40)$$

For $m, p \to \infty$, $\chi_{DL} \to \frac{N}{K} = \frac{N}{N-T}$. This is the optimal download cost.

3) Decoding complexity: Since the decoding requires computing the sum of the received results, the decoding complexity of the proposed scheme is negligible. This is an important advantage of the proposed scheme compared to existing polynomial coding schemes in the literature, which require polynomial interpolation.

E. Comparison with other schemes

There exists a trade-off between the upload cost and the download cost depending on the kind of partitioning employed for matrix multiplication. In [25], [29], the trade-off within the class of schemes that employ row-by-column partitioning is considered. The other class of schemes that employ column-by-row partitioning, like MatDot and our scheme, provide different points on the upload cost-download cost trade-off. A comparative summary is described in Table I. The GASP [18], secure PolyDot [24], and USCSA [25] schemes, which employ row-by-column partitioning generally result in a higher upload cost and lower download cost than those employing column-by-row partitioning. Consider K row-wise partitions of $\mathbf{A}$, and K column-wise partitions of $\mathbf{B}$. For $T = 0$, the matrices are encoded by evaluating their corresponding polynomials at $N = K^2$ distinct points in $\mathbb{F}_q$. SDMM schemes employing the row-by-column partitioning require $N = O(K^2)$ servers. Hence, the upload cost is $\chi_{UL} = \frac{N}{K} = O(\frac{N}{\sqrt{N}}) = O(\sqrt{N})$. Since the complexity of evaluating a polynomial at N points is $O(N \log^2 N \log\log N)$ [30], the total encoding complexity is $O(\frac{mn}{\sqrt{N}} N \log^2 N \log\log N) \approx O(mn\sqrt{N} \log^2 N \log\log N)$. This is significantly larger than the encoding complexity of our scheme, which grows only logarithmically with N.

The secure MatDot scheme of [24], which employs column-by-row partitioning, has an upload cost of $\chi_{UL} = \frac{2N}{N-2T+1}$, which is also larger than that of our scheme by a factor of two (see Fig. 2).

Compared to [26], which also considers the general case with the source nodes separate from the user, our scheme provides significant improvements in terms of the upload cost. The upload cost achieved by the scheme in [26] is given by $\chi_{UL} = \frac{\min(2K^2 + 2T - 3, K^2 + KT + T - 2)}{K}$, which is significantly higher than that provided by our scheme (see Fig. 2). We also highlight that, as opposed to the scheme in [26], our scheme does not require inter-server communication in the computation and communication phases for multiplying two matrices, which significantly reduces both the latency and the complexity.

IV. STRAGGLER MITIGATION

The FFT-based scheme in the previous sections does not provide robustness against straggling servers. We need all
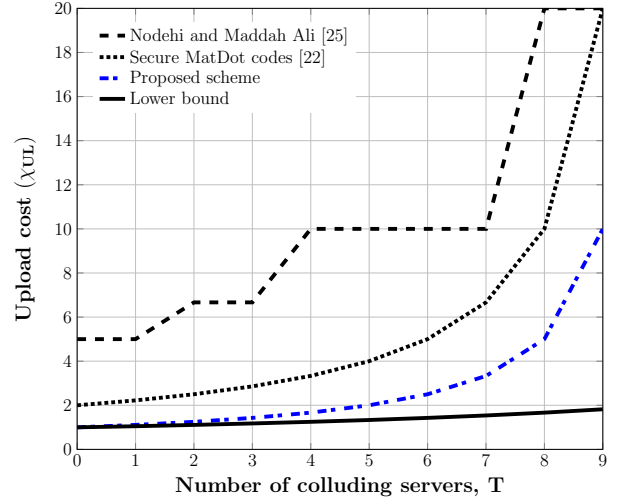

Fig. 2: Comparison of the upload cost of SDMM of two matrices for different number of colluding servers out of a total of $N = 20$ servers. Only the values of $T \le 9$ are considered since the proposed scheme is defined only for $T < N/2$. The maximum value of K is calculated for each T.

the evaluations from all the servers to recover the desired result. In this section we present an extension of our FFT-based scheme, which provides a certain level of robustness to straggling servers by incorporating both column-wise and row-wise partitioning of the matrices.

When providing straggler mitigation, typically the goal is to minimize the *recovery threshold*, which refers to the minimum integer r such that the computation result can be recovered from any r responsive servers. Hence, instead of specifying the number of servers, and minimizing the upload cost, in this section, we consider arbitrary partitioning of the matrices, and identify the corresponding upload cost and the recovery threshold.

In particular, we employ row-wise partitioning of $\mathbf{A}$ (and column-wise for $\mathbf{B}$) to introduce straggler-robustness. That is, $\mathbf{A}$ is partitioned into a $K_2 \times K_1$ array of equal-sized blocks, and $\mathbf{B}$ is partitioned into a $K_1 \times K_3$ array of equal-sized blocks. The idea is that the secret shares of each row of $\mathbf{A}$ (and each column of $\mathbf{B}$) are generated, while polynomially-coded shares of each column of $\mathbf{A}$ (and each row of $\mathbf{B}$) are generated for straggler robustness. The proposed scheme provides a recovery threshold of $N - \left(\lceil \frac{N}{K_1 + 2T} \rceil - K_2 K_3\right)$ in general, and $N - \left(\lceil \frac{N}{K_1 + T} \rceil - K_2 K_3\right)$ for computing with own data.

The matrices $\mathbf{A}$ and $\mathbf{B}$ are partitioned into $K_1 K_2$ and $K_2 K_3$ blocks, denoted by $\mathbf{A}_{i,j}, i \in [K_2], j \in [K_1]$ and $\mathbf{B}_{j,k}, j \in [K_1], k \in [K_3]$, respectively. Matrices $\mathbf{R}_{i,j}, i \in [K_2], j \in [T]$, and $\mathbf{S}_{j,k}, j \in [T], k \in [K_3]$, are generated, whose entries are i.i.d. uniform random variables from $\mathbb{F}_q$. The matrices $\mathbf{R}$ and $\mathbf{S}$, defined as

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{1,1} & \cdots & \mathbf{R}_{1,T} \\ \vdots & \ddots & \vdots \\ \mathbf{R}_{K_2,1} & \cdots & \mathbf{R}_{K_2,T} \end{bmatrix}, \qquad (41)$$

This article has been accepted for publication in IEEE Transactions on Information Theory. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIT.2022.3158868

9

Scheme	Upload cost	Download cost	Encoding complexity	Decoding complexity
Secure MatDot [24]	$\frac{2N}{N-2T+1}$	$N = 2(K+T)-1$	$O(\frac{mn}{K} N \log^2 N \log \log N)$	$O(\frac{mp}{K} N \log^2 N \log \log N)$
GASP [18], Secure PolyDot [24], USCSA [25], Nodehi et al. [26]	$O(\sqrt{N})$	$\frac{N}{N-O(T)}$	$O(\frac{mn}{\sqrt{N}} N \log^2 N \log \log N)$	$O(\frac{mp}{\sqrt{N}} N \log^2 N \log \log N)$
Proposed scheme	$\frac{N}{N-2T}$	$N = K + 2T$, if $n \geq K \min(m,p)$ $\frac{Nn(m+p-n/K)}{Kmp}$, if $n \geq \min(m,p)$ $\frac{N(m+p-n/K)}{K(m+p-n)}$, if $n \leq \min(m,p)$	$O(\frac{mnN}{N-2T} \log N)$	$O(1)$
Proposed scheme using 1 inter-server comm. round (Sec. III-C)	$\frac{N}{N-2T}$	$\frac{N}{N-T}$	$O(\frac{mnN}{N-2T} \log N)$	$O(\frac{mp}{N-T} N \log N)$

TABLE I: Comparative summary of different schemes.

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{1,1} & \cdots & \mathbf{S}_{1,K_3} \\ \vdots & \ddots & \vdots \\ \mathbf{S}_{T,1} & \cdots & \mathbf{S}_{T,K_3} \end{bmatrix}, \quad (42)$$

are used as secret keys to encode the matrices $\mathbf{A}$ and $\mathbf{B}$, respectively, by first appending them in the following manner:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{R} \end{bmatrix}, \quad (43)$$

$$\tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{B} \\ \mathbf{S} \end{bmatrix}, \quad (44)$$

and then constructing the following multivariate polynomials:

$$\mathbf{A}(x_1, x_2) = \sum_{i=1}^{K_2} \sum_{j=1}^{K_1} \mathbf{A}_{i,j} x_2^{i-1} x_1^{j-1}$$
$$+ \sum_{i=1}^{K_2} \sum_{j=1}^{T} \mathbf{R}_{i,j} x_2^{i-1} x_1^{K_1+j-1} \quad (45)$$

$$\mathbf{B}(x_1, x_2) = \sum_{j=1}^{K_1} \sum_{k=1}^{K_3} \mathbf{B}_{j,k} x_1^{-(j-1)} x_2^{(k-1)K_2}$$
$$+ \sum_{j=1}^{T} \sum_{k=1}^{K_3} \mathbf{S}_{j,k} x_1^{-(K_1+T+j-1)} x_2^{(k-1)K_2}. \quad (46)$$

The product polynomial $\mathbf{C}(x_1, x_2)$ is given by

$$\mathbf{C}(x_1, x_2) = \sum_{i=1}^{K_2} \sum_{k=1}^{K_3} \left(\sum_{j=1}^{K_1} \mathbf{A}_{i,j} \mathbf{B}_{j,k} \right) x_2^{K_2(k-1)+i-1}$$
$$+ \text{(terms with non-zero powers of } x_1). \quad (47)$$

A. Sharing phase

Let there be a total of $N \geq N_1 N_2$ servers, where $N_1 = K_1 + 2T$ and $N_2 \geq K_2 K_3$. The user evaluates polynomials $\mathbf{A}(x_1, x_2)$ and $\mathbf{B}(x_1, x_2)$ at points $(x_1, x_2) = (\alpha_{N_1}^r, \beta_s), r = 0, \ldots, N_1 - 1, s = 0, \ldots, N_2 - 1$, where α_{N_1} is a primitive N_1^{th} root of unity in $\mathbb{F}_q$, and $\beta_s, \forall s \in [N_2]$ are distinct points in $\mathbb{F}_q$.

The user sends $\mathbf{A}(\alpha_{N_1}^r, \beta_s)$ and $\mathbf{B}(\alpha_{N_1}^r, \beta_s)$ to server i, where $i = (s-1)N_1 + r - 1, \forall r \in [N_1], \forall s \in [N_2]$.

B. Computation phase

Server i, where $i = (s-1)N_1 + r - 1$, computes $\mathbf{C}(\alpha_{N_1}^r, \beta_s) = \mathbf{A}(\alpha_{N_1}^r, \beta_s)\mathbf{B}(\alpha_{N_1}^r, \beta_s)$, and sends this share to the user as soon as it is computed.

C. Reconstruction phase

The user collects and computes the averages of all the evaluations of $\mathbf{C}(x_1, x_2)$ on the points having the same x_2 coordinate, which removes the terms with non-zero exponents of x_1. Therefore, for every $x_2 \in [N_2]$, the user obtains

$$\mathbf{f}(x_2) = \frac{1}{N_1} \sum_{i=0}^{N_1-1} \mathbf{C}(\alpha_{N_1}^i, x_2) \quad (48)$$
$$= \sum_{i=1}^{K_2} \sum_{k=1}^{K_3} \left(\sum_{j=1}^{K_1} \mathbf{A}_{i,j} \mathbf{B}_{j,k} \right) x_2^{K_2(k-1)+i-1}. \quad (49)$$

The user then interpolates the polynomial $\mathbf{f}(x_2)$ from any $K_2 K_3$ evaluations to obtain $\sum_{j=1}^{K_1} \mathbf{A}_{i,j} \mathbf{B}_{j,k}$ for all $(i,k) \in [K_2] \times [K_3]$, therefore obtaining the final result $\mathbf{C} = \mathbf{AB}$.

D. Special case: computation with own data

For the special case when the data matrices belong to the user, the encoding polynomials are constructed by the user as follows.

$$\mathbf{A}(x_1, x_2) = \sum_{i=1}^{K_2} \sum_{j=1}^{K_1} \mathbf{A}_{i,j} x_2^{i-1} x_1^{j-1} + \sum_{i=1}^{K_2} \sum_{j=1}^{T} \mathbf{R}_{i,j} x_2^{i-1} x_1^{K_1+j-1}$$
$$(50)$$

$$\mathbf{B}(x_1, x_2) = \sum_{j=1}^{K_1} \sum_{k=1}^{K_3} \mathbf{B}_{j,k} x_1^{-(j-1)} x_2^{K_2(k-1)}$$
$$+ \sum_{j=1}^{T} \sum_{k=1}^{K_3} \mathbf{S}_{j,k} x_1^{-(K_1+j-1)} x_2^{K_2(k-1)}. \quad (51)$$

By this construction, the matrices $\mathbf{R}$ and $\mathbf{S}$ also appear in the constant coefficient of the polynomials $\mathbf{f}(x_2)$, and the user must pre-compute the products of the private random matrices in order to obtain the desired result.

Remark 5. *Note that, to reconstruct the desired result, the user requires all N_1 evaluations on the points (x_1, x_2) for $x_1 = 1, \alpha_{N_1}, \ldots, \alpha_{N_1}^{N_1-1}$ and constant x_2. In other words, for any $K_2 K_3$ values of s, the results from all the groups of servers $[(s-1)N_1 : sN_1 - 1]$ are necessary and sufficient for reconstructing the desired result. Therefore, this scheme provides a slightly weaker notion of 'group-wise' recovery threshold, that is, the minimum integer r such that any r groups of servers as defined above are sufficient for recovering the computation result. This represents the best-case recovery threshold. The proposed scheme provides a group-recovery*

threshold of $K_2K_3(K_1+2T)$ in general, and $K_2K_3(K_1+T)$ for computation with own data.

Remark 6. *If at least $r = N - \left(\lceil \frac{N}{K_1+2T} \rceil - K_2K_3\right)$ number of servers are responsive, that is, there is at most one unresponsive server in each of $\left(\lceil \frac{N}{K_1+2T} \rceil - K_2K_3\right)$ groups of $K_1 + 2T$ servers, while all the remaining servers are responsive, then it is guaranteed that K_2K_3 groups of K_1+2T servers, as defined in Remark 5, are responsive. Therefore, the recovery threshold is given by $r = N - \left(\lceil \frac{N}{K_1+2T} \rceil - K_2K_3\right)$. However, this is the worst-case recovery threshold, and it is often likely that the required number of groups are responsive with a smaller total number of responsive servers. The average recovery threshold would lie between the group-recovery threshold and the recovery threshold.*

E. Performance Analysis:

- **Upload Cost:** We have $\chi_{UL} = \frac{N(K_3mn+K_2np)}{K_1K_2K_3(mn+np)}$.
- **Encoding complexity:** The complexity of encoding matrix $\mathbf{A}$ for N servers is $O\left(\frac{mn}{K_1K_2}\left(N_1 \log N_1\right)\left(N_2 \log^2 N_2 \log\log N_2\right)\right)$. The encoding complexity for matrix $\mathbf{B}$ can be computed similarly.
- **Decoding complexity:** The decoding requires addition, which has negligible complexity compared to multiplication in a finite field, followed by interpolation of a polynomial with K_2K_3 terms, which has complexity $O(n \log^2 n \log\log n)$ for $n = K_2K_3$ [30]. Note that the decoding complexity of secure PolyDot scheme in [24] is given by the complexity of interpolating a polynomial with $K_2K_3(K_1+T)+K_2K_1+K_2T-1$ terms, while our scheme requires interpolating a polynomial with K_2K_3 terms, which is an order of magnitude smaller.
- **Field size:** The field must have the N_1^{th} roots of unity, and must be big enough to have K_2K_3 distinct elements. Therefore, $N_1|q-1$ and $q > K_2K_3$ must be satisfied.

V. SECURE DISTRIBUTED MULTIPLICATION OF MULTIPLE MATRICES

In this section, we consider the multiplication of multiple matrices. Such computations, known as matrix chain multiplications, occur in many applications in signal processing, graph theory, and network analysis. We extend our scheme proposed in Section III to implement multiplication of multiple matrices, that is, given Γ source nodes generating the matrices $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(\Gamma)}$, the user wants to obtain the product $\mathbf{C} = G(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(\Gamma)}) = \mathbf{A}^{(1)}\cdots\mathbf{A}^{(\Gamma)}$ securely from distributed computation over N available servers. A naive method would be for the servers to securely compute the multiplication of two matrices at a time using the proposed scheme in Section III, and send the results of the computations to the user so that he reconstructs the intermediate computation results, that is, the product of a subset of the matrices, before re-encoding the intermediate result and sending its secret shares to the servers to multiply with the next matrix. However, such a naive method incurs an unnecessary amount of communication cost

between the user and the servers, and also requires the user to re-encode the intermediate matrices multiple times, thus increasing the latency of the computation. The naive method also leaks information about the intermediate computations to the user. Next, we propose a more efficient alternative, that also satisfies the user privacy constraint.

The following scheme proceeds iteratively in multiple rounds by obtaining the shares of $\mathbf{C}^{(\gamma)} \triangleq \mathbf{A}^{(1)}\cdots\mathbf{A}^{(\gamma)}$, denoted by $[[\mathbf{C}^{(\gamma)}]]_i \triangleq [[\mathbf{A}^{(1)}\cdots\mathbf{A}^{(\gamma)}]]_i, \forall i \in [N]$, in the $(\gamma-1)^{th}$ round, for $\gamma \in [\Gamma]$.

A. Sharing phase

For $K = N - 2T$, the $(N, N-2T, T)$ left-shares $[[\mathbf{A}^{(1)}]]_i^L$ and right-shares $[[\mathbf{A}^{(\gamma)}]]_i^R$, $\forall\gamma \in [2:\Gamma]$, are sent to server i.

B. Computation phase

In the computation phase of the γth round, where $\gamma \in [\Gamma-1]$, server $i \in [N]$ computes the shares $[[\mathbf{H}^{(\gamma+1)}]]_i = [[\mathbf{C}^{(\gamma)}]]_i^L[[\mathbf{A}^{(\gamma+1)}]]_i^R$, $\forall\gamma \in [\Gamma-1]$. The secret shares $[[\mathbf{H}^{(\gamma+1)}]]_i$ are evaluations on the Nth roots of unity of a polynomial $\mathbf{H}^{(\gamma+1)}(x)$, which is similar to the product polynomial obtained in Eq. (11), and whose constant term is the matrix $\mathbf{C}^{(\gamma+1)} = \mathbf{C}^{(\gamma)}\mathbf{A}^{(\gamma+1)}$, while the remaining terms are uniformly distributed random matrices.

C. Communication phase

In the communication phase, the servers exchange shares of their results from the computation phase in a secure way, similarly to [26], to convert their secret shares $[[\mathbf{H}^{(\gamma+1)}]]$ to $(N, N-2T, T)$ left-shares of matrix $\mathbf{C}^{(\gamma+1)}$. To do this,

- Server i generates $(N, N-2T, T)$ left-shares of $[[\mathbf{H}^{(\gamma+1)}]]_i$, evaluated on the Nth roots of unity $\alpha_N^{j-1}, \forall j \in [N]$, with α_N being a primitive N^{th} root of unity in $\mathbb{F}_q$, and enumerated as $[[\mathbf{H}^{(\gamma+1)}]]_{i,j}^L = \left[[[[\mathbf{H}^{(\gamma+1)}]]_i]\right]_j^L, \forall j \in [N]$.
- Server i sends the left-share $[[\mathbf{H}^{(\gamma+1)}]]_{i,j}^L$ to server j. The privacy requirement against the servers is satisfied, since any T colluding servers cannot gain any information about server i's share $[[\mathbf{H}^{(\gamma+1)}]]_i$ from the left-shares received in the communication phase.
- Server j averages the received left-shares $[[\mathbf{H}^{(\gamma+1)}]]_{i,j}^L, \forall i \in [N]$, to obtain the $(N, N-2T, T)$ left-share $[[\mathbf{C}^{(\gamma+1)}]]_j^L$.

To see the correctness of the above procedure, note that, for given N, K and T values, the secret sharing scheme is linear for both left and right shares; that is, $[[\mathbf{A}]]+[[\mathbf{B}]] = [[\mathbf{A}+\mathbf{B}]]$. Therefore, following from Eq. (18), we have

$$\frac{1}{N}\sum_{i=1}^{N}[[\mathbf{H}^{(\gamma+1)}]]_{i,j}^L = \frac{1}{N}\sum_{i=1}^{N}\left[\left[[[\mathbf{H}^{(\gamma+1)}]]_i\right]_j^L \quad (52)$$

$$= \left[\left[\mathbf{C}^{(\gamma+1)}\right]\right]_j^L. \quad (53)$$

The scheme proceeds in a recursive manner, looping back to the computation phase for the $(\gamma + 1)^{th}$ round.

D. Reconstruction phase

At the end of the $(\Gamma - 1)^{th}$ round, the servers have access to the secret shares of the matrix $\mathbf{C}^{(\Gamma)} = \mathbf{A}^{(1)} \cdots \mathbf{A}^{(\Gamma)}$. The servers send the shares $[[\mathbf{C}^{(\Gamma)}]]_i, \forall i \in [N]$, to the user, which then computes the average of these shares to obtain

$$\frac{1}{N} \sum_{i=1}^{N} [[\mathbf{C}^{(\Gamma)}]]_i = \mathbf{A}^{(1)} \cdots \mathbf{A}^{(\Gamma)}. \tag{54}$$

E. Performance analysis

Upload cost: Each server receives one $(N, N-2T, T)$ share of all $\mathbf{A}^{(\gamma)}, \gamma \in [\Gamma]$. Thus we have

$$\chi_{UL} = \frac{\sum_{i=1}^{N} \sum_{\gamma=1}^{\Gamma} \frac{H([[\mathbf{A}^{(\gamma)}]]_i)}{K}}{\sum_{\gamma=1}^{\Gamma} H(\mathbf{A}^{(\gamma)})} \tag{55}$$

$$= \frac{N}{N - 2T}. \tag{56}$$

Encoding complexity: Since the encoding involves the computation of FFT, the complexity is $O(m_\gamma m'_\gamma \frac{N}{K} \log N)$ for encoding the matrix $\mathbf{A}^{(\gamma)}, \forall \gamma \in [\Gamma]$.

Complexity of the communication phase: The complexity of server i generating $(N, N-2T, T)$ shares of $[[\mathbf{H}^{(\gamma+1)}]]_i$ is $O(m_\gamma m'_{\gamma+1} \frac{N}{K} \log N)$, while the complexity of server j averaging the received shares from the other servers is ignored as it requires only addition. The cost of inter-server communication is $\frac{N-1}{N-2T}$ per server, since each server communicates $N-1$ shares to the remaining servers.

Decoding complexity: The partial decoding in the communication phase of each intermediate round, as well as the final decoding of the result after the completion of the $(\Gamma - 1)^{th}$ round requires only the addition of the received results from all the servers. Therefore, the decoding complexity is negligible.

Security against the user: The scheme constructed in this section preserves security against the user, since the intermediate computation results are not communicated to the user in any form. In contrast, this security constraint is violated by the naive scheme described at the start of this section, because the user obtains the result of each intermediate computation.

VI. Secure matrix algebra

In this section, we describe algorithms for performing matrix operations besides matrix multiplication that are useful in matrix algebra. Some operations, like matrix inversion, can be reduced to matrix multiplication, and can be implemented with the SDMM schemes described in the preceding sections.

A. Matrix addition and multiplication by a scalar

Addition and scalar multiplication follow easily from the linear nature of the secret sharing scheme. We have $[[\mathbf{A} + \mathbf{B}]] = [[\mathbf{A}]] + [[\mathbf{B}]]$, and $[[c\mathbf{A}]] = c[[\mathbf{A}]]$, where $c \in \mathbb{F}_q$. For addition, the shares must both be either left-shares or right-shares.

B. Changing from (N, K_1, T_1) left-shares to (N, K_2, T_2) right-shares

Suppose the servers store (N, K_1, T_1) left-shares of matrix $\mathbf{A}$. The goal is to let the servers obtain (N, K_2, T_2) right-shares of matrix $\mathbf{A}$. A special case of this algorithm with $K_1 = 1$, $K_2 = N - 2T$ and $T_1 = T_2 = T$ was used earlier in the communication phase of Section V. The procedure takes the following steps (see Fig. 3):

1) Server i generates (N, K_2, T_2) right-shares of $[[\mathbf{A}]]_i^L$ evaluated on the Nth roots of unity $\alpha_N^{j-1}, \forall j \in [N]$, and enumerated as $[[\mathbf{A}]]_{i,j}^{LR} = \left[\left[[[\mathbf{A}]]_i^L\right]\right]_j^R, \forall j \in [N]$.
2) Server $i \in [N]$ sends the right-share $[[\mathbf{A}]]_{i,j}^{LR}$ to server j.
3) Server j interpolates the received shares $[[\mathbf{A}]]_{i,j}^{LR}, i = 1, \ldots, N$, using inverse FFT (IFFT) to obtain a polynomial whose first K_1 coefficients are K_1 column-wise partitions of the share $[[\mathbf{A}]]_j^R$. Stacking them column-wise gives the share $[[\mathbf{A}]]_j^R$.

A procedure to convert right-shares to left-shares can be obtained similarly to the above procedure. For example, to convert right-shares to left-shares, step 1 of the above procedure generates (N, K_2, T_2) left-shares of $[[\mathbf{A}]]_i^R$, enumerated as $[[\mathbf{A}]]_{i,j}^{RL} = \left[\left[[[\mathbf{A}]]_i^R\right]\right]_j^L, \forall j \in [N]$.

To see the correctness of the above procedure, note that IFFT of a sequence of (N, K_1, T_1) left-shares $\left\{[[\mathbf{A}]]_1^L, \ldots, [[\mathbf{A}]]_N^L\right\}$ gives the sequence $\{\mathbf{A}_1, \ldots, \mathbf{A}_{K_1}, \mathbf{R}_1, \ldots, \mathbf{R}_{T_1}\}$. Similarly, IFFT of a sequence of (N, K_1, T_1) right-shares $\left\{[[\mathbf{A}]]_1^R, \ldots, [[\mathbf{A}]]_N^R\right\}$ gives the sequence $\{\mathbf{A}_1, \ldots, \mathbf{A}_{K_1}, \mathbf{0}_1, \ldots, \mathbf{0}_{T_1}, \mathbf{R}_1, \ldots, \mathbf{R}_{T_1}\}$. Since each coefficient of the IFFT sequence is a linear combination of the elements of the input sequence; for all $l \in [K_1]$, the lth coefficient of the IFFT, denoted by $IFFT_l(.)$, of the sequence of shares received by server j is given by

$$IFFT_l\left([[\mathbf{A}]]_{1,j}^{LR}, \ldots, [[\mathbf{A}]]_{N,j}^{LR}\right) \tag{57}$$

$$= IFFT_l\left(\left[\left[[[\mathbf{A}]]_1^L\right]\right]_j^R, \ldots, \left[\left[[[\mathbf{A}]]_N^L\right]\right]_j^R\right) \tag{58}$$

$$= \left[\left[IFFT_l\left([[\mathbf{A}]]_1^L, \ldots, [[\mathbf{A}]]_N^L\right)\right]\right]_j^R \tag{59}$$

$$= [[\mathbf{A}_l]]_j^R, \tag{60}$$

where $\mathbf{A}_l, l \in [K_1]$ is the lth column-wise partition used for obtaining the original (N, K_1, T_1) left-shares of matrix $\mathbf{A}$. Thus, server j obtains the secret shares $[[\mathbf{A}_l]]_j^R, l = 1, \ldots, K_1$.

Remark 7. *For $K_1 \geq 2$, left-shares cannot be directly converted to left-shares using the above procedure. Similarly, right-shares cannot be directly converted to right-shares. However, $(N, 1, T_1)$ left-shares can be directly converted to both (N, K_2, T_2) left-shares and right-shares using the above procedure. For converting $(N, 1, T_1)$ left-shares to (N, K_2, T_2) left-shares, step 1 of the above procedure generates (N, K_2, T_2) left-shares of $[[\mathbf{A}]]_i^L$, enumerated as $[[\mathbf{A}]]_{i,j}^{LL} = \left[\left[[[\mathbf{A}]]_i^L\right]\right]_j^L, \forall j \in [N]$.*

C. Transpose of a matrix

Consider that server i stores a (N, K_1, T_1) left-share of $\mathbf{A}, i \in [N]$. The goal is to obtain a procedure, through which
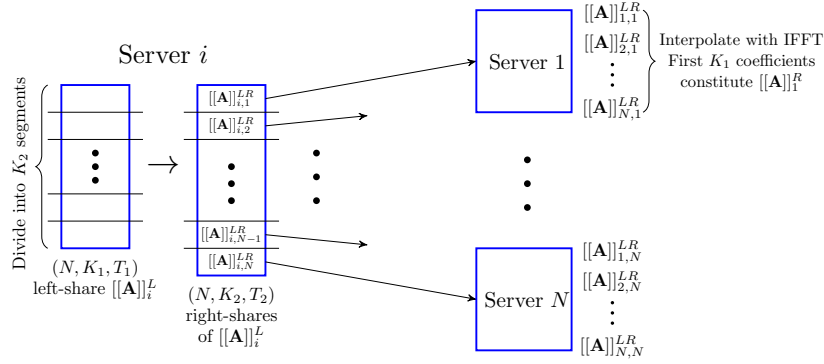
Fig. 3: Conversion from left-share to right-share (right-share to left-share conversion can be done similarly).

the servers end up with the (N, K_2, T_2) left-shares of $\mathbf{A}^{tr}$ instead. The procedure takes the following steps:

1) Server $i \in [N]$ performs the transpose operation on the left-share $[[\mathbf{A}]]_i^L$, to obtain a share $[[\mathbf{A}^{tr}]]_i = ([[\mathbf{A}]]_i^L)^{tr}$. We have

$$[[\mathbf{A}^{tr}]]_i = ([[\mathbf{A}]]_i^L)^{tr} \tag{61}$$

$$= \sum_{l=1}^{K_1} \mathbf{A}_l^{tr} \alpha_N^{(i-1)(l-1)} + \sum_{l=1}^{T_1} \mathbf{R}_l^{tr} \alpha_N^{(i-1)(K+l-1)}, \tag{62}$$

where $\mathbf{A}_l^{tr}, l = 1, \ldots, K_1$ are equivalent to row-wise partitions of $\mathbf{A}^{tr}$. Note that $[[\mathbf{A}^{tr}]]_i$ is neither a left-share nor a right-share, but is equivalent to having a row-wise partitioning of $\mathbf{A}^{tr}$ employed in right-encoding, and having the exponents of the secret keys employed in left-encoding.

2) Server i then generates (N, K_2, T_2) left-shares of $[[\mathbf{A}^{tr}]]_i$ evaluated on the Nth roots of unity $\alpha_N^{j-1}, \forall j \in [N]$, and enumerated as $[[\mathbf{A}^{tr}]]_{i,j}^L = \left[[[[\mathbf{A}^{tr}]]_i] \right]_j^L, \forall j \in [N]$.

3) Server i sends the left-share $[[\mathbf{A}^{tr}]]_{i,j}^L$ to server j.

4) Server j interpolates the received shares $[[\mathbf{A}^{tr}]]_{i,j}^L, i = 1, \ldots, N$, using IFFT to obtain the first K_1 coefficients, which are K_1 row-wise partitions of the share $[[\mathbf{A}^{tr}]]_j^L$. Stacking them row-wise gives the share $[[\mathbf{A}^{tr}]]_j^L$.

D. Exponentiation

Suppose the user wants to compute $G(\mathbf{A}) = \mathbf{A}^r, r \in \mathbb{N}$. $(N, N - 2T, T)$ left-shares of matrix $\mathbf{A}$ are sent to the servers. The servers then implement the scheme to generate $(N, N - 2T, T)$ right-shares of matrix $\mathbf{A}$ as described in Section VI-B. Thus, the servers now have the shares of matrix $\mathbf{A}$ for left and right multiplication. The servers then implement the scheme for multiple matrix multiplication described in Section V. If $r = 2^n$, the computation can be done in $\log r$ rounds of computation and communication phases by computing $\mathbf{A}^2, \mathbf{A}^4, \mathbf{A}^8, \ldots, \mathbf{A}^r$ in successive rounds. If $r \neq 2^n$, then consider the binary expansion $\{a_{B-1}, \ldots, a_0\}$ of r, such that

$r = \sum_{i=0}^{B-1} a_i 2^i$, where B is the maximum number of bits required to represent r. Then, the desired result is given by

$$\mathbf{A}^r = \mathbf{A}^{\sum_{i=0}^{B-1} a_i 2^i} \tag{63}$$

$$= \prod_{i=0}^{B-1} \mathbf{A}^{a_i 2^i}. \tag{64}$$

If the Hamming weight of the binary expansion of r is h, then the computation requires $h - 1$ extra rounds of computation and communication phases to perform the computation of Eq. (64), resulting in a total of $\log r + h - 1$ rounds.

Upload cost: $(N, N - 2T, T)$ left-shares of matrix $\mathbf{A}$ are uploaded by the source, while the $(N, N - 2T, T)$ right-shares of matrix $\mathbf{A}$ are generated in-situ by the share-conversion algorithm, therefore not required to be sent by the source. Therefore, we have $\chi_{UL} = \frac{N}{N - 2T}$.

E. Solving the linear system $\mathbf{AX} = \mathbf{B}$ with secure Gaussian elimination

The linear system $\mathbf{AX} = \mathbf{B}$, where the elements of $\mathbf{A}$ and $\mathbf{B}$ belong to $\mathbb{F}_q$, can be solved by performing Gaussian elimination (GE) on the augmented matrix $(\mathbf{A}|\mathbf{B})$. Setting $\mathbf{B}$ equal to the identity matrix, the solution of the linear system also gives the matrix inverse $\mathbf{A}^{-1}$, if it exists. The GE method performs elementary row operations on the augmented matrix of the linear system, and row interchanges, also called pivoting, to transform the linear system into its row-echelon form. A scheme for secure GE is described in [5], which takes element-wise secret shares of matrices $\mathbf{A}$ and $\mathbf{B}$ as inputs, and outputs the solution of the linear system. Computing element-wise secret shares of a matrix is equivalent to computing $(N, 1, T)$ secret shares of matrices $\mathbf{A}$ and $\mathbf{B}$. Therefore, if the servers store (N, K, T) secret shares of matrix $\mathbf{A}$ and $\mathbf{B}$, they must first be converted to $(N, 1, T)$ secret shares using the scheme in Section VI-B.

F. Secure matrix inversion

Besides solving the linear system $\mathbf{AX} = \mathbf{I}$ to compute the matrix inverse, as described in the previous section, a different procedure, inspired from that in [6], is described below, through which the servers start from (N, K, T) right-shares of a square matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$, and end up with (N, K, T) left-shares of $\mathbf{A}^{-1}$ instead.

1) A uniformly distributed random matrix $\boldsymbol{\Phi} \in \mathbb{F}_q^{m \times m}$, which is used as the secret key, is secretly shared with the servers. We assume that there is no central entity that can generate and share the secret key securely with the servers. The shares of the secret key are generated by the servers in a decentralized manner as follows: For $i \in [N]$, server i generates a random matrix $\boldsymbol{\Phi}^{(i)} \in \mathbb{F}_q^{m \times m}$, and generates its (N, K, T) left-shares evaluated on the Nth roots of unity $\alpha_N^{j-1}, \forall j \in [N]$. Server i sends the left-share $[[\boldsymbol{\Phi}^{(i)}]]_j^L$ to server $j \in [N] \setminus \{i\}$. From the left-shares received by server j, it computes the left-share $[[\boldsymbol{\Phi}]]_j^L = [[\boldsymbol{\Phi}^{(1)} + \cdots + \boldsymbol{\Phi}^{(N)}]]_j^L = [[\boldsymbol{\Phi}^{(1)}]]_j^L + \cdots + [[\boldsymbol{\Phi}^{(N)}]]_j^L$. Thus, each server obtains a left-share of a common secret key $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{(1)} + \cdots + \boldsymbol{\Phi}^{(N)}$.

2) Server j securely computes $[[\mathbf{P}]]_j = [[\boldsymbol{\Phi}\mathbf{A}]]_j = [[\boldsymbol{\Phi}]]_j^L[[\mathbf{A}]]_j^R$.

3) The servers reconstruct matrix $\mathbf{P}$ from its secret shares by exchanging their secret shares of matrix $\mathbf{P}$ with every other server, and averaging the received shares, similar to Eq. (18). Thus, each server obtains the public matrix $\mathbf{P} = \boldsymbol{\Phi}\mathbf{A}$. The servers gain no information of the matrix $\mathbf{A}$ from the matrix $\mathbf{P}$, therefore satisfying the privacy constraint against the servers.

4) Each server computes the matrix inverse $\mathbf{P}^{-1} = \mathbf{A}^{-1}\boldsymbol{\Phi}^{-1}$.

5) For $j = 1, \ldots, N$, server j then obtains a left-share of the inverse of matrix $\mathbf{A}$ as follows: $[[\mathbf{A}^{-1}]]_j^L = \mathbf{P}^{-1}[[\boldsymbol{\Phi}]]_j^L = [[\mathbf{P}^{-1}\boldsymbol{\Phi}]]_j^L$.

6) The servers can now perform further computation on the left-shares obtained, or deliver their left-shares to the user, who then performs IFFT on the received results to obtain the matrix $\mathbf{A}^{-1}$.

Upload cost: $(N, N-2T, T)$ shares of matrix $\mathbf{A}$ are uploaded by the source for Step 2, where SDMM of random matrix $\boldsymbol{\Phi}$ and input matrix $\mathbf{A}$ is performed, thus incurring an upload cost of $\chi_{UL} = \frac{N}{N-2T}$.

G. Iterative matrix inversion

The method for secure matrix inversion introduced above includes an intermediate step (step 4) that involves the inversion of a secure full-size matrix at each server. While the procedure satisfies the privacy requirements, it may contradict with the motivation of distributed computation. Iterative matrix inversion algorithms, for example Newton's method [31], do not involve direct matrix inversions, but instead proceed with matrix multiplications, and therefore, are amenable to efficient distributed implementation. Newton's method, however, provides only an approximation of the matrix inverse. Newton's method for inverting matrices is derived from Newton's method for finding the root of a function. The procedure is as follows [31]:

- Set $f(\mathbf{X}) = \mathbf{A} - \mathbf{X}^{-1}$. Note that the root of $f(\mathbf{X})$ is $\mathbf{A}^{-1}$. Apply Newton's method for finding its root, as follows:

$$\mathbf{X}_{i+1} = \mu_i \mathbf{X}_i \left(2\mathbf{I} - \mathbf{A}\mathbf{X}_i\right), \qquad (65)$$

where $\mu_i = 1$ for all $i > 0$.

- **Choosing the initial estimate $\mathbf{X}_0$:** Quadratic convergence is obtained if $||\mathbf{A}\mathbf{X}_0 - \mathbf{I}|| < 1$. This is satisfied if $\mathbf{X}_0 = \mu_o \mathbf{A}^T$ is picked as the initial estimate, with the value of μ_0 as proposed in [31].

The matrix addition and multiplication operations can be performed securely in a distributed manner using the algorithms described in this paper.

H. Computation of arbitrary polynomials:

The algorithms for securely performing matrix addition, transpose, exponentiation, inverse and multiplication that have been described in this paper allow the user to compute arbitrary matrix polynomials on distributed servers. For example, a function of the following form,

$$G(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3) = \mathbf{A}_1^2\mathbf{A}_2 + c\mathbf{A}_3^{-1}, \qquad (66)$$

can be computed securely on distributed servers as follows: first, the servers securely compute $\mathbf{A}_1^2\mathbf{A}_2$ using the scheme for multiple matrix multiplication; then, the servers add the shares $[[\mathbf{A}_1^2\mathbf{A}_2]]$ to the shares $[[c\mathbf{A}_3^{-1}]]$, computed using one of the secure matrix inversion methods described, to finally obtain the secret shares of the final result, which the user receives and decodes to obtain the desired result.

I. Secure learning from local datasets

Consider that data from D source nodes, each with a different size of dataset, is used for training a fully connected deep neural network. In a fully connected neural network, the input layer of neurons performs the matrix multiplication $\mathbf{W}\mathbf{X}$, where $\mathbf{W}$ is the weight matrix associated with the layer of neurons, and $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_D)$, where $\mathbf{X}_i, i \in [D]$ is the dataset belonging to source node i. SDMM schemes in [16], [18], [25], [26], based on column-wise partitioning of matrix $\mathbf{X}$, code across different data points, thus requiring the local datasets of the sources to be encoded at a central location, which leads to a privacy concern. In contrast, the SDMM algorithm proposed in this paper, based on row-wise partitioning of matrix $\mathbf{X}$, encodes each dataset independently. We have

$$[[\mathbf{X}]]^R = \left([[\mathbf{X}_1]]^R, \ldots, [[\mathbf{X}_D]]^R\right), \qquad (67)$$

that is, each source can deliver the right shares of its dataset to the servers independent of other sources.

Linear regression- Consider the computation of the MMSE estimate in a linear regression problem, where the optimum estimate is given by $\beta = \left(\mathbf{X}^{tr}\mathbf{X}\right)^{-1}\mathbf{X}^{tr}\mathbf{Y}$. The local datasets $(\mathbf{X}_i, \mathbf{Y}_i)$ can be delivered to the servers similarly to Eq. (67), and then the left-shares and right-shares of $\mathbf{X}^{tr}$ and $\mathbf{Y}$ can be generated by using the algorithms to convert shares and perform secure transpose described in this section.

J. Achieving the optimal upload and download cost for SDMM

If we assume that the cost of inter-server communication is negligible compared to the upload and download costs, optimal communication costs can be achieved simultaneously for both upload and download while ignoring the inter-server

communication costs. The source nodes upload $(N, N-T, T)$ shares of the input matrices to the servers, resulting in an upload cost of $\chi_{UL} = \frac{N}{N-T}$, which is shown to be a lower bound on the upload cost for SDMM in [25]. Using the algorithm proposed in Section VI-B, the $(N, N-T, T)$ shares can then be converted to $(N, N-2T, T)$ shares for implementing the SDMM algorithm proposed in this paper. The secret shares of the computation results are then converted to $(N, N-T, T)$ shares using the algorithm in Sections III-C and VI-B. These shares are sent to the user, resulting in a download cost of $\chi_{DL} = \frac{N}{N-T}$, which is known to be the optimal download cost for SDMM [16]. This procedure circumvents the trade-off between the upload cost and the download cost for SDMM schemes, studied earlier in [25].

The assumption that the cost of inter-server communication is negligible is justified in many practical scenarios involving computing clusters, where the computing servers are connected with high-speed communication links, while the links between the source nodes and the servers, and between the user and the servers may have limited bandwidth. However, when the inter-server communication costs (delay, bandwidth and/or energy) are non-negligible, the extra rounds of communication among the servers, required for various share conversions, become prohibitive.

VII. CONCLUSION AND DISCUSSION

In this paper we developed a novel polynomial coded computation scheme achieving a near-optimal performance in terms of the upload cost for SDMM across N servers, any T of which may collude. We also proposed a scheme achieving the optimal upload cost for the special case when the user requesting the computation is also the source of the matrices to be computed upon. The scheme involves evaluating the constructed polynomials at the roots of unity in an appropriate finite field, which is equivalent to taking the discrete Fourier transform of the constituent matrices. The encoding and decoding complexity is also lower than all the other schemes in the literature. For a special case of the data matrices having certain asymptotic dimensions, our scheme also achieves the optimal download cost. We also introduced a method for straggler mitigation, which provides group-wise tolerance to straggling servers. Straggler tolerance is achieved at the expense of an increase in the upload cost. We further extended our scheme to implement multiplication of multiple matrices, while keeping the input matrices and all the intermediate computations secure against any T colluding servers, with a minimal upload cost. This presents a substantial improvement in performance in terms of the upload cost for multiplication of multiple matrices over existing schemes in the literature. Moreover, we described procedures for other common matrix operations, some of which can be reduced to a set of matrix multiplications, thus allowing us to compute arbitrary matrix polynomials.

For future work, methods for securely performing other matrix operations, such as matrix decompositions, on distributed servers will be explored. It would also be interesting to develop schemes for multiplication of arbitrary number of matrices with minimum inter-server communication. From a more practical perspective, an interesting problem to look at is the simultaneous scheduling of the computation and communication phases to minimize the overall latency.

REFERENCES

[1] S. Rane, W. Sun, and A. Vetro, "Secure function evaluation based on secret sharing and homomorphic encryption," in *Annual Allerton Conf. on Comm., Control, and Computing*, Sep. 2009, pp. 827–834.

[2] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Secure statistical analysis using rlwe-based homomorphic encryption," in *Information Security and Privacy*, E. Foo and D. Stebila, Eds. Cham: Springer International Publishing, 2015, pp. 471–487.

[3] D. H. Duong, P. K. Mishra, and M. Yasuda, "Efficient secure matrix multiplication over lwe-based homomorphic encryption," *Tatra Mountains Mathematical Publications*, vol. 67, no. 1, pp. 69 – 83, 2016. [Online]. Available: https://content.sciendo.com/view/journals/tmmp/67/1/article-p69.xml

[4] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Practical packing method in somewhat homomorphic encryption," in *Data Privacy Management and Autonomous Spontaneous Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 34–50.

[5] N. J. Bouman and N. de Vreede, "New protocols for secure linear algebra: Pivoting-free elimination and fast block-recursive matrix decomposition," *IACR Cryptology ePrint Archive*, vol. 2018, p. 703, 2018.

[6] R. Cramer and I. Damgård, "Secure distributed linear algebra in a constant number of rounds," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 119–136.

[7] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, March 2018.

[8] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3368–3376. [Online]. Available: http://proceedings.mlr.press/v70/tandon17a.html

[9] 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*. Curran Associates, Inc., 2017, pp. 4403–4413.

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

[11] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1585–1589, 2018.

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

[13] 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 the Twenty-Second International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and M. Sugiyama, Eds., vol. 89. PMLR, 16–18 Apr 2019, pp. 1215–1225. [Online]. Available: https://proceedings.mlr.press/v89/yu19b.html

[14] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960. [Online]. Available: https://doi.org/10.1137/0108018

[15] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for straggler exploitation with heterogeneous workers," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 251–256.

[16] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

[17] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176

[18] R. G. L. D'Oliveira, S. El Rouayheb, and D. Karpuk, "GASP codes for secure distributed matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 7, pp. 4038–4050, 2020.

[19] Z. Jia and S. A. Jafar, "On the capacity of secure distributed batch matrix multiplication," *IEEE Transactions on Information Theory*, vol. 67, no. 11, pp. 7420–7437, 2021.

[20] K. Banawan and S. Ulukus, "Multi-message private information retrieval: Capacity results and near-optimal schemes," *IEEE Transactions on Information Theory*, vol. 64, no. 10, pp. 6842–6862, Oct 2018.

[21] R. Tajeddine, O. W. Gnilke, D. Karpuk, R. Freij-Hollanti, and C. Hollanti, "Private information retrieval from coded storage systems with colluding, Byzantine, and unresponsive servers," *IEEE Transactions on Information Theory*, vol. 65, no. 6, pp. 3898–3906, June 2019.

[22] Q. Yu and A. S. Avestimehr, "Entangled polynomial codes for secure, private, and batch distributed matrix multiplication: Breaking the "cubic" barrier," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 245–250.

[23] Z. Chen, Z. Jia, Z. Wang, and S. A. Jafar, "GCSA codes with noise alignment for secure coded multi-party batch matrix multiplication," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, jun 2020. [Online]. Available: https://doi.org/10.1109%2Fisit44484.2020.9174230

[24] M. Aliasgari, O. Simeone, and J. Kliewer, "Distributed and private coded matrix computation with flexible communication load," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 1092–1096.

[25] J. Kakar, A. Khristoforov, S. Ebadifar, and A. Sezgin, "Uplink cost adjustable schemes in secure distributed matrix multiplication," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 1124–1129.

[26] H. A. Nodehi and M. A. Maddah-Ali, "Limited-sharing multi-party computation for massive matrix operations," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1231–1235.

[27] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*. New York, NY, USA: Association for Computing Machinery, 2019, p. 351–371. [Online]. Available: https://doi.org/10.1145/3335741.3335756

[28] S. Lin, T. Y. Al-Naffouri, Y. S. Han, and W. Chung, "Novel polynomial basis with fast Fourier transform and its application to Reed-Solomon erasure codes," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6284–6299, 2016.

[29] J. Kakar, S. Ebadifar, and A. Sezgin, "Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation," *ArXiv*, vol. abs/1810.13006, 2018.

[30] A. Borodin and R. Moenck, "Fast modular transforms," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 366 – 386, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000074800292

[31] M. Ylinen, A. Burian, and J. Takala, "Direct versus iterative methods for fixed-point implementation of matrix inversion," in *IEEE Int'l Symposium on Circuits and Systems*, vol. 3, 2004, pp. III–225.

Cong Ling received the B.S. and M.S. degrees in electrical engineering from the Nanjing Institute of Communications Engineering, Nanjing, China, in 1995 and 1997, respectively, and the Ph.D. degree in electrical engineering from the Nanyang Technological University, Singapore, in 2005.

He is currently a Reader in the Electrical and Electronic Engineering Department at Imperial College London. His research interests are coding, information theory, and security, with a focus on lattices. Dr. Ling has served as an Associate Editor of IEEE Transactions on Communications and of IEEE Transactions on Vehicular Technology.

Deniz Gündüz [S'03-M'08-SM'13] received the B.S. degree in electrical and electronics engineering from METU, Turkey in 2002, and the M.S. and Ph.D. degrees in electrical engineering from NYU Tandon School of Engineering (formerly Polytechnic University) in 2004 and 2007, respectively. After his PhD, he served as a postdoctoral research associate at Princeton University, as a consulting assistant professor at Stanford University, and as a research associate at CTTC in Barcelona, Spain. ln Sep. 2012, he joined the Electrical and Electronic Engineering Department of Imperial College London, UK, where he is currently a Professor of Information Processing, and serves as the deputy head of the Intelligent Systems and Networks Group. He is also a part-time faculty member at the University of Modena and Reggio Emilia, Italy, and has held visiting positions at University of Padova (2018-2020) and Princeton University (2009-2012).

His research interests lie in the areas of communications and information theory, machine learning, and privacy. Dr. Gündüz is a Distinguished Lecturer for the IEEE Information Theory Society (2020-22). He is an Area Editor for the IEEE Transactions on Information Theory, IEEE Transactions on Communications, and the IEEE Journal on Selected Areas in Communications (JSAC) - Special Series on Machine Learning in Communications and Networks. He also serves as an Editor of the IEEE Transactions on Wireless Communications. He is the recipient of the IEEE Communications Society - Communication Theory Technical Committee (CTTC) Early Achievement Award in 2017, a Starting Grant of the European Research Council (ERC) in 2016, and several best paper awards.

Nitish Mital received the B.Tech and M.Tech degrees in electrical engineering with specialization in communications and signal processing from Indian Institute of Technology Bombay in 2015, and his PhD degree in Electrical and Electronics Engineering from Imperial College London in 2020. He is currently a postdoctoral research associate in the Intelligent Systems and Networks Group in the Electrical and Electronics Engineering department of Imperial College London. He was the recipient of a H2020 Marie-Sklodowska-Curie fellowship from 2016 to 2019. During his PhD in 2018, he was a visiting researcher in the NYU Tandon School of Engineering.

His research interests lie in the areas of communications and information theory, distributed computing and security, and machine learning for data compression and communications. Dr. Mital received the Best Student Paper Award at the 2018 IEEE Wireless Communications and Networking Conference (WCNC).